Elsevier Editorial System(tm) for Information and Software Technology
Manuscript Draft

Manuscript Number:

Title: Obsolete Software Requirements

Article Type: Research paper

Corresponding Author: Mr. Krzysztof Wnuk, MSC

Corresponding Author's Institution: Lund University

First Author: Krzysztof Wnuk, Doctoral Candidate

Order of Authors: Krzysztof Wnuk, Doctoral Candidate; Tony Gorschek, Professor; Showayb Zahda, Master of Science

Abstract: [Context] Coping with rapid change of requirements is crucial for staying competitive in software business. Frequently changing customer needs and fierce competition are example reasons for quick evolution of requirements that often become obsolete even before project completion. [Objective] Although the phenomenon of obsolete requirements and the implications of not handling them are known, there is a lack of empirical research dedicated to understanding the nature of obsolete software requirements and their impact and role in requirements management.[Method] In this paper, we report results from an empirical investigation with 219 respondents from different companies, aimed at investigating the phenomenon of obsolete software requirements. [Results] Our results contain, but are not limited to, defining the phenomenon of obsolete software requirements, investigating how they are handled in industry today, and their potential impact. [Conclusion] Supported by our results we conclude that obsolete software requirements constitute a significant challenge for companies developing software intensive products, in particular in large projects, and that the companies rarely have processes of handling obsolete software requirements. Further, our results call for future research in creating automated methods for obsolete software requirements identification and management, that could enable efficient management of obsolete software requirements in large projects.

Obsolete Software Requirements

Krzysztof Wnuk

Department of Computer Science, Lund University, Ole Römers väg 3, SE-223 63 Lund, Sweden
Tel.: +46-46 222 45 17
Fax: +46-46 13 10 21
email: Krzysztof.Wnuk@cs.lth.se

Tony Gorschek

School of Computing Software Engineering Research Lab, Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden
Tel.: +46 455-38 58 17
Fax: +46 455-38 50 57
email: Tony.Gorschek@bth.se

Showayb Zahda

School of Computing Software Engineering Research Lab, Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden
Tel.: +46 455-38 58 17
Fax: +46 455-38 50 57
email: shzc10@student.bth.se

Abstract

[Context] Coping with rapid change of requirements is crucial for staying competitive in software business. Frequently changing customer needs and fierce competition are example reasons for quick evolution of requirements that often become obsolete even before project completion. [Objective] Although the phenomenon of obsolete requirements and the implications of not handling them are known, there is a lack of empirical research dedicated to understanding the nature of obsolete software requirements and their impact and role in requirements management. [Method] In this paper, we report results from an empirical investigation with 219 respondents from different companies, aimed at investigating the phenomenon of obsolete software requirements. [Results] Our results contain, but are not limited to, defining the phenomenon of obsolete software requirements, investigating how they

are handled in industry today, and their potential impact. [Conclusion] Supported by our results we conclude that obsolete software requirements constitute a significant challenge for companies developing software intensive products, in particular in large projects, and that the companies rarely have processes of handling obsolete software requirements. Further, our results call for future research in creating automated methods for obsolete software requirements identification and management, that could enable efficient management of obsolete software requirements in large projects.

Keywords: requirements management, obsolete requirements, survey, empirical study

---

1. Introduction

Software, as a business, is a demanding environment where a growing number of users, rapid introduction of new technologies, and fierce competition are inevitable [1, 2, 3]. This rapidly changing business environment is challenging traditional requirements engineering (RE) approaches [4, 5, 6]. The major challenges in this environment are high volatility and quick evolution of requirements, requirements that often tend to become obsolete even before project completion [1, 7, 8, 9]. At the same time the product release time is crucial [10, 11, 12] for the success of the software products, especially in emerging or rapidly changing markets [10].

Coping with rapid change of requirements is crucial as time-to-market pressures often make early pre-defined requirements specifications inappropriate almost immediately after their creation [7]. In Market-Driven Requirements Engineering (MDRE) the pace of incoming requirements [2], and requirements change, is high, and software companies have to identify which requirements may become obsolete or outdated. Rapid identification and handling of potentially obsolete requirements is important as large volumes of degrading requirements threatens the requirements management. In extreme cases, obsolete requirements could dramatically extend project timelines, increase the total cost of the project or even cause project failure - and even the successful realization of the obsolete requirements adds little or no product value [13, 14, 15]. Thus, the identification, handling, and removal of obsolete requirements is central.

The phenomenon of obsolete requirements and the implications of not handling them are known, see e.g. [16, 13, 14, 17, 18, 19, 20]. However, very

little research into requirements management or guidelines, see e.g. [21, 22, 23, 24, 25, 26], standards [27, 28], explicitly mentioning the phenomenon of Obsolete Software Requirements (OSRs) has been performed. The term itself is only partly defined and empirically anchored [17].

In this paper, we present results from an empirical study, based on a survey with 219 respondents from different companies, aimed at investigating the phenomenon of obsolete requirements. This included, but was not limited to, defining the phenomenon based on the perceptions of practitioners in industry. Further, the study also aims to collect data on how obsolete requirements are perceived, their impact, and how they are handled in industry today.

This paper is structured as follows: Section 2 provides the background and related work. Section 3 describes the research methodology. Section 4 describes and discusses the results of the study. Section 5 concludes the paper.

2. Background and Related Work

Requirements management, as an integral part of requirement engineering [9, 24], takes care of the data created in requirements elicitation and development phases of the project and integrates this data into the overall project flow [9]. Also, it supports later lifecycle modification of the requirements [9]. As changes occur during the entire software project lifetime [29], managing changes to the requirements is a main concern of requirements management [22, 23] for large software systems. Moreover, in some contexts like Market-Driven Requirements Engineering (MDRE) a constant stream of new requirements and change requests is inevitable [2]. Uncontrolled changes to software may cause cost of the regression testing exceeding 100000 dollars [9]. As pointed out by Hood [9], the absence of requirements management may sooner or later cause outdated requirements specifications as the information about the changes to original requirements is not fed back to the requirements engineers. Moreover, requirements management process descriptions in literature seldom contain managing obsolete requirements [22, 21].

Requirements creep and requirements leakage (also referred as uncontrolled requirements creep) [30, 31] is related to OSRs. Scope creep has also been mentioned as having a big impact on risk and risk management in enterprise data warehouse projects [32]. Moreover, DeMarco and Lister also listed scope creep as one of the five core risks during the requirements phase and

state that the risk is a direct indictment of how requirements were gathered in the first place [33]. Scope creep may lead to significant scope reductions, which in turn postpone the implementation of the planned functionality and may in the end cause requirements becoming obsolete [8].

Despite its importance as a concept, in relation to managing requirements for software products, the phenomenon of Obsolete Software Requirements (OSRs) seems to be underrepresented in literature. To the best of our knowledge, only a handful of articles and books mention the term obsolete requirements or features. Among the existing evidence, Loesch and Ploederefer [18] claim that the explosion of the number of variable features and variants in a software product line context is partly caused by the fact that obsolete variable features are not removed. Murphy and Rooney [13] stress that requirements have 'a shelf life' and suggest that the longer it takes from the requirements definition to implementation, the higher the risk of change is. Moreover, they also state that change makes requirements obsolete as well as that obsolete requirements can dramatically extend project timelines and increase the total cost of the project. Similarly, Stephen et al. [14] list obsolete requirements as one of the symptoms of failure of IT project for UK government. Albeit the report does not define obsolete requirements *per see*, the mentioned symptom of failure (obsolete requirements) is caused by inability to unlock the potential offered by new technologies by timely responding to the rapid pace of change in technology (the mentioned inability is ascribed to obsolete requirements).

The phenomenon of Obsolete Software Requirements has not yet been mentioned by standardization bodies in software engineering. Neither the IEEE 830 standard [27] nor CMMI (v.1.3) [28] mention obsolete software requirements as a phenomenon, nor are actions, processes or techniques suggested in relation to handling the complexity. On the other hand, Savolainen et al. [17] propose a classification of atomic product line requirements into: non-reusable, mandatory, variable and obsolete. Moreover they propose a short definition of obsolete requirements and the process of managing these requirements for software product lines ("by marking them obsolete and hence not available for selection into subsequent systems"). Mannion et al. [19] propose a category of variable requirements called obsolete and suggest dealing with them as described by Savolainen et al. [17].

Obsolete requirements are related to the concept of requirements volatility. SWEBOOK classifies requirements into a number of dimensions where one of them is volatility and stability, mentioning that some volatile re-

quirements may render obsolete [34]. On the other hand Nurmuliani and Zowghi [35] propose a taxonomy of requirements change where one of the reasons for requirements changes is obsolete functionality, defined as "functionality that is no longer required for the current release or has no value for the potential users". For this paper, we understand requirements volatility as a factor that influences the change of requirements and is different from the obsolescence of requirements. Obsolescence of requirements is defined as a situation were volatile become outdated and remain in the requirements databases [36, 37].

Looking at previous work, the obsolescence of software artefacts has been mentioned in the context of obsolete hardware and electronics (for example in military, avionics or other industries). Among others, Herald et al. propose a system of obsolescence management framework for system components (in this case hardware, software, constraints) that is mainly concerned with system design and evolution phases [20]. Albeit the framework contains a technology roadmapping component, it does not explicitly mention OSRs. Merola [15] describes the software obsolescence problem in today's defence systems of systems (in the software components level, also called COTS), and stresses that even though the issue has been recognized of equal gravity as hardware obsolescence issue, it has not reached the same level of visibility. Merola outlines some options for managing software obsolescence, e.g. negotiated with the vendor downgrading the software license, using wrappers and software application programming interfaces or performing market analysis and surveys of software vendors.

Due to a limited number of studies in the literature that are dedicated to the phenomenon of OSRs, we have decided to investigate the concept utilizing a survey research strategy. We investigate to what extent obsolete software requirements are: 1) perceived as a real phenomenon in industry, 2) perceived as a real problem in industry. Moreover, we investigate how OSRs are identified and managed in practice, and what contextual factors influence OSRs.

3. Research methodology

This section covers the research questions, the research methodology, and data collection methods used in the study.

3.1. Research questions

Due to a limited number of related empirical studies identified in relation to OSRs, we decided to mainly focus on understanding the phenomenon of OSR, and its place in the requirements engineering landscape. Thus existence, descriptive, as well as classification questions dominate the research questions outlined in Table 1, complemented with aim [38]. Throughout the research questions, we have used the following definition of Obsolete Software Requirements (OSRs), based on the literature study and the survey:
"An obsolete software requirement is a software requirement (implemented or not) that is no longer required for the current release or future releases, and it has no value (business goals) for the potential customers or users of a software product for various reasons.[1] *"*

3.2. Research design

A survey was chosen as the main tool to collect empirical data. A survey enabled reaching a larger number of respondents from geographically diverse locations [39] as well as automation of data collection and analysis, flexibility and convenience to both researchers and participants, [38, 40, 41].

As the goal of the survey was to elicit as much information from industry practitioners as possible in relation to OSRs, we opted for an inclusive approach to catch as many answers as possible. This prompted the use of convenience sampling [41]. The details in relation to survey design and data collection are outlined below.

3.2.1. Survey design

The questionnaire was created based on a literature review of relevant topics, such as requirements management, volatility, and requirements traceability (see Section 2). The questions were developed iteratively; each version of the questionnaire was discussed among the authors and evaluated in relation to how well the questions reflect the research questions and the research goals. The resulting final questionnaire is available online [42].

The questionnaire contained 15 open and closed questions of different format i.e. multiple choice questions or single choice questions. In case of open questions, respondents had the possibility to provide their own answer

[1]For reader convenience we present the definition in this section, rather than post results. The description of how the definition was derived is available in Section 4.

Table 1: Research questions

Research question	**Aim**	**Example answer**
RQ1: Based on empirical data, what would be an appropriate definition of Obsolete Software Requirements (OSR)?	Instead of defining the phenomenon ourselves we base the definition on how the phenomenon is perceived in industry.	"An obsolete software requirements is a requirement that has not been included into the scope of the project for the last 5 projects"
RQ2: What is the impact of the phenomenon of obsolete software requirements on the industry practice?	To investigate to what degree is OSR a serious **concern**.	"Yes it is somehow serious"
RQ3: Does requirement type affect the likelihood of a software requirement becoming obsolete?	Are there certain types of requirements that become obsolete more often than others, can these types be identified?	"A market requirement will become obsolete much faster than a legal requirement."
RQ4: What methods exist, in industry practice, that help to identify obsolete software requirements?	To enact a process to detect/identify/find obsolete software requirements or nominate requirements that risk becoming obsolete.	"To read the requirements specification carefully and check if any requirements are obsolete"
RQ5: When OSRs are identified, how are they typically handled in industry?	In order to identify possible alternatives for OSR handling, we first need to understand how they are handled today.	"We should mark found obsolete requirements as obsolete but keep them in the requirements database"
RQ6: What context factors, such as project size or domain, influence OSRs?	As a step in understanding and devising solutions for handling OSRs, it is important to identify contextual factors that have an influence on the phenomenon of obsolete requirements.	"OSRs are more present in a large projects and for products that are sold to an open market (MDRE context)"
RQ7: Where in the requirements life cycle should OSRs be handled?	To position requirements obsolescence in the requirements engineering life cycle.	"They should be a part of the requirements traceability task"

as well as select a pre-defined answer from the list. For closed questions, we used a Likert scale from 1 to 5, where 1 corresponds to *Not likely* and 5 to *Very likely* [43].

The questionnaire was divided into two parts, one related to OSRs (9 questions), and one to demographics (6 questions). Table 2 below shows the survey questions, with a short description of their purpose (2nd column), the list of relevant references (3rd column), and a link to what research question it primarily addresses (4th column). For reasons of brevity we do not present the entire survey in the paper. However, the complete survey questionnaire, including the references that were used to construct the categories for the answers is available online [42].

3.2.2. Operation (execution of the survey)

The survey was conducted using a web-survey support website called SurveyMonkey [45]. Invitations to participate in the questionnaire were sent to the potential audience via:

- Personal emails - utilizing the contact networks of the authors
- Social network website [46] - placing the link to the questionnaire on the board of the SE and RE groups and contacting individuals from the discovered groups based on their designated titles e.g. senior software engineer, requirements engineer, system analyst, project manager, only to name a few
- Mailing lists - requirements engineering and software engineering discussion groups [47]
- Software companies and requirements management tools' vendors [48]

Master students and undergraduate students were excluded as potential respondents as their experience was judged as insufficient to answer the questionnaire. The questionnaire was published online on the 3rd of April 2011 and the data collection phase ended after 30 days on the 3rd of May 2011. In total, about 1700 individual invitations were sent out. The number of complete collected responses was 219. The response rate was around 8%, which is an acceptable level [38, 39]. The results of the survey are presented in Section 4.

Table 2: Mapping between the questionnaire questions and the research questions

Question	**Purpose**	**Relevant references**	**RQ**
Q1	To derive the definition of the Obsolete Software Requirements	[17, 20, 35, 15]	RQ1
Q2	To investigate the impact of the OSRs on the industry practice	[13, 14]	RQ2
Q3	To investigate how likely the various types of requirements would become obsolete.	The list of requirements types was derived from analyzing several requirements classifications [36, 37]	RQ3
Q4	To investigate the possible ways of identifying OSR in the requirements documents	[18, 20]	RQ4
Q5	To investigate the possible actions to be taken against obsolete requirements after they are discovered	[18, 20]	RQ5
Q6	To investigate if large or smaller projects are more or less affected by OSRs	The classification of different sizes of requirements engineering was adopted from Regnell et al. [44]	RQ6
Q7	To investigate if OSRs are related to the software context	[14]	RQ6
Q8	To understand where in the requirements life cycle should OSRs be handled	Current standards for requirements engineering and process models [27, 28] do not consider obsolete requirements.	RQ5, partly RQ7
Q9	To investigate if industry has any process of managing OSR	[18, 17]	RQ5

3.3. Validity

In this section, we discuss the threats to validity in relation to research design and data collection phases. The four perspectives on validity discussed in this section are based on classification proposed by Wohlin et al. [49].

Construct validity. The construct validity is concerned with the relation between the observations from the study and the theories behind the research. The way how questions were phrased is one of the threats to construct validity. This threat was alleviated by revising the questionnaire by the authors of this paper and an independent reviewer who is a native English speaker and writer. In order to minimize the risk of misunderstanding or misinterpreting the survey questions by respondents, a pilot study was conducted on master students in software engineering. Still, the reader should keep in mind that the data given by respondents is not based on any objective measurements and thus its subjectivity affects the interpretability of the results. The mono-operational bias [49] threat to construct validity is addressed by collecting data from more than 200 respondents from 45 countries. Finally, the mono-method bias [49] threat to construct validity was partly addressed by analyzing related publications. Albeit several related publications have been identified (see Section 2), this threat is not fully alleviated and requires further studies. Finally, considering social threats to construct validity it is important to mention the evaluation apprehension threat [49]. The respondents' anonymity was guaranteed.

Conclusion validity. The conclusion validity is concerned with the ability to draw correct conclusions from the study. In order to address the reliability of measures threat, the questions used in the study were reviewed by the authors of this paper and one external reviewer, a native English speaker. The low statistical power threat [49] was addressed by using as suitable statistical test as it was possible on the given type of data. Before running the tests, it was tested if the assumptions of the particular test were not violated. However, since multiple tests were conducted on the same data, the risk of type-I error increases and using for example the Bonferroni correction should be discussed here. Since the correction was criticized by a number of authors [50, 51] it remains an open question if it should be used. Therefore, we report the p-values of all performed tests in case the readers want to evaluate the results using the Bonferroni correction or other adjustment techniques [50]. Finally, the random heterogeneity of subjects [49] threat should be mentioned here as this aspect was only partly controlled. However, this low heterogeneity of subjects allows us to state conclusions of a

greater external validity.

Internal validity. Threats to internal validity are related to factors that affect the causal relationship between the treatment and the outcome. The instrumentation threat [49] to internal validity was addressed by reviews of the questionnaire and the pilot study. The maturation threat to internal validity was alleviated by measuring the time needed to participate in the survey in the pilot study (15 minutes). The selection bias threat to internal validity is relevant as non-random sampling was used. Since the respondents were volunteers, their performance may vary from the performance of the whole population [49]. However, the fact that 219 participants from 45 countries with different experience and industrial roles answered the survey minimizes the effect of this threat.

External validity. The threats to external validity concern the ability to generalize the result of research efforts to industrial practice [49]. The survey research method was selected to assure as many responses as possible, generating more general results [38, 52, 41] than a qualitative interview study. Moreover, the large number of respondents from various countries, contexts and professions contributes to generalizability of results.

4. Results and Analysis

The survey was answered by 219 respondents. In case of questions that enabled multiple answers, we calculated the results over the total number of answers, not respondents. For questions that used a Likert scale, we present the results using average rating and the percentage received by each answer on the scale. The results are presented in percentage form and complemented by the number of answers or respondents (when relevant). Statistical analysis (where relevant) was performed using chi-square test [53], and the complete results from the analysis are listed online, including contingency tables for some of the answers [54].

4.1. Demographics

Figure 1 depicts the top 10 respondent countries (out of 45)[2]. The full list of the countries is available in [55]. The US and the UK constitute about

[2]The actual category names have been changed for readability purposes. The original names are mentioned using *italics* in the paper and are available in the survey questionnaire [42]

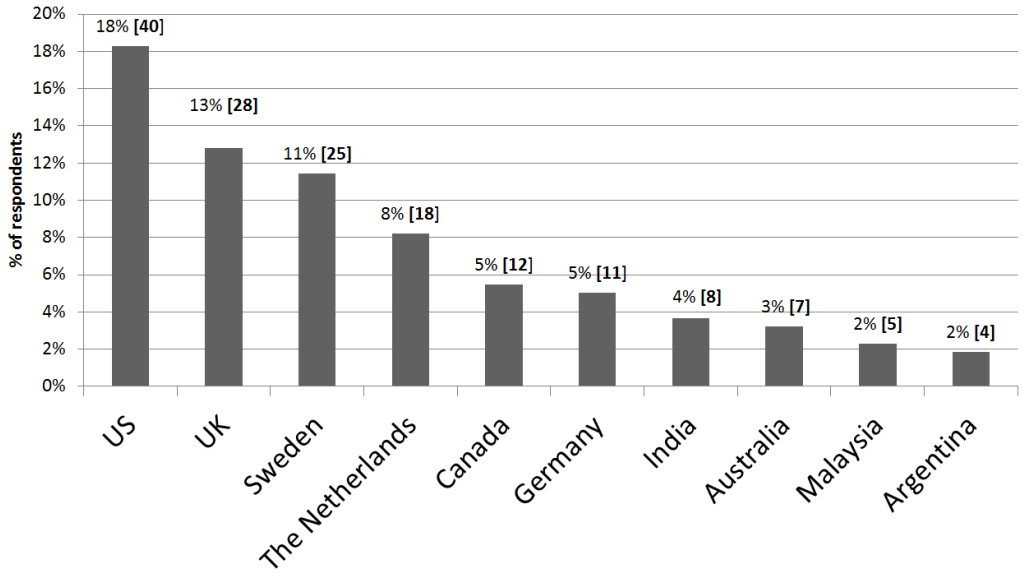

Figure 1: Top 10 countries among respondents

30% of the total respondents. 54% of the respondents came from Europe.

The main roles of the respondents in their organization can be seen in Figure 2. About one quarter of the respondents (24.9% or 54 respondents) described their role as requirements engineers, analysts or coordinators.

The second biggest category, *Other* (with 30 answers), include roles such as *System Engineers, Software Quality Assurance, Process Engineers, Business Analysts*. The third biggest category was *Researchers* or *Academics* (11.5% of all answers). *Software project managers* and *Software Architect or Designer* roles got the same number of respondents (22 respondents each). Twelve respondents declared their main role as *Software Product Manager*, which is a relatively high number, as product managers are generally few in an organization. This would seem to indicate that middle and senior managers overall represented a substantial part of the respondents.

Looking at the type of the business domain of the respondents, Figure 3 gives an overview. A total of 32.8% stated the *IT or Computer and Software Services*. The second largest group of answers is *Engineering* e.g. automotive, aerospace, energy (12.5%). These were followed by *Telecommunication* (10.7%) and *Consultancy* (9.3%).

The sizes of the respondents' organizations are depicted in Figure 4. We

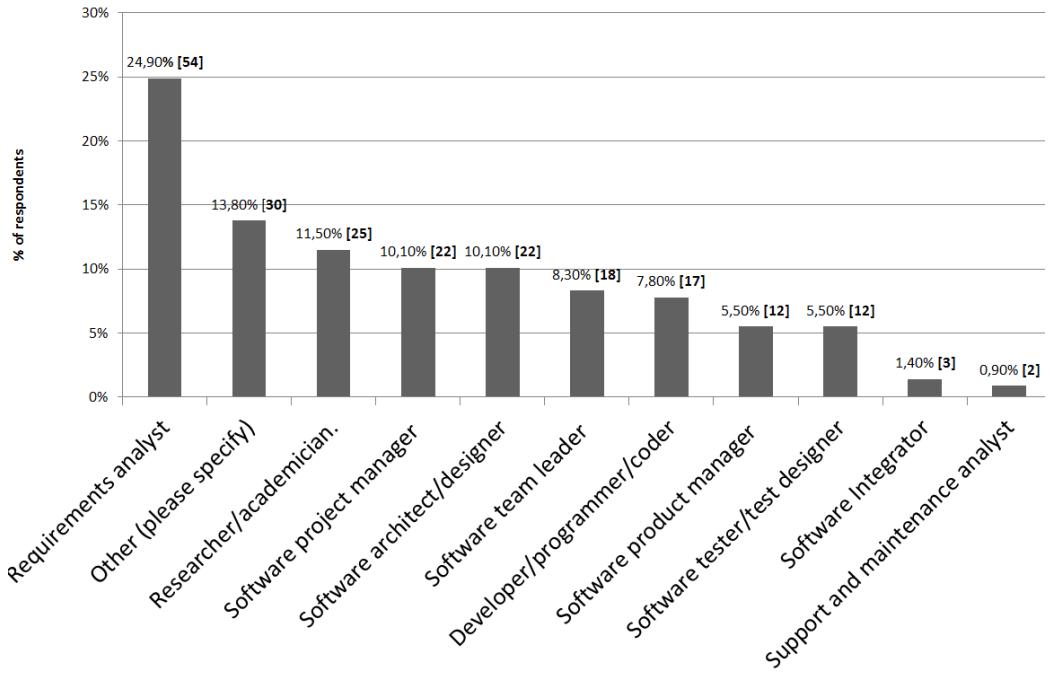

Figure 2: Main role of respondents

can see that more than half of the respondents work in large companies (>501 employees).

Looking at the average duration of typical project in the respondents' organizations (see Figure 5), about half of the respondents (∼45%) were involved in projects that lasted for less than a year and a quarter of the respondents were involved in projects that lasted between one and two years. One quarter indicated projects typically lasting more than two years.

The development methodologies and processes used by the respondents were also investigated. Since this question allowed for the possibility of providing multiple answers, the results are calculated over the number of responses. The results are depicted in Figure 6. Agile development tops the list of answers with approximately a quarter (23.6%), and in the second place we find incremental and evolutionary methodology (18.8%). Surprisingly, waterfall is still common and widely used (17.7%). In the *Other* category, most of the respondents reported that they mixed several methodologies, or that they had their own tailored methodology.

The last question in the demographics part investigates the type of re-

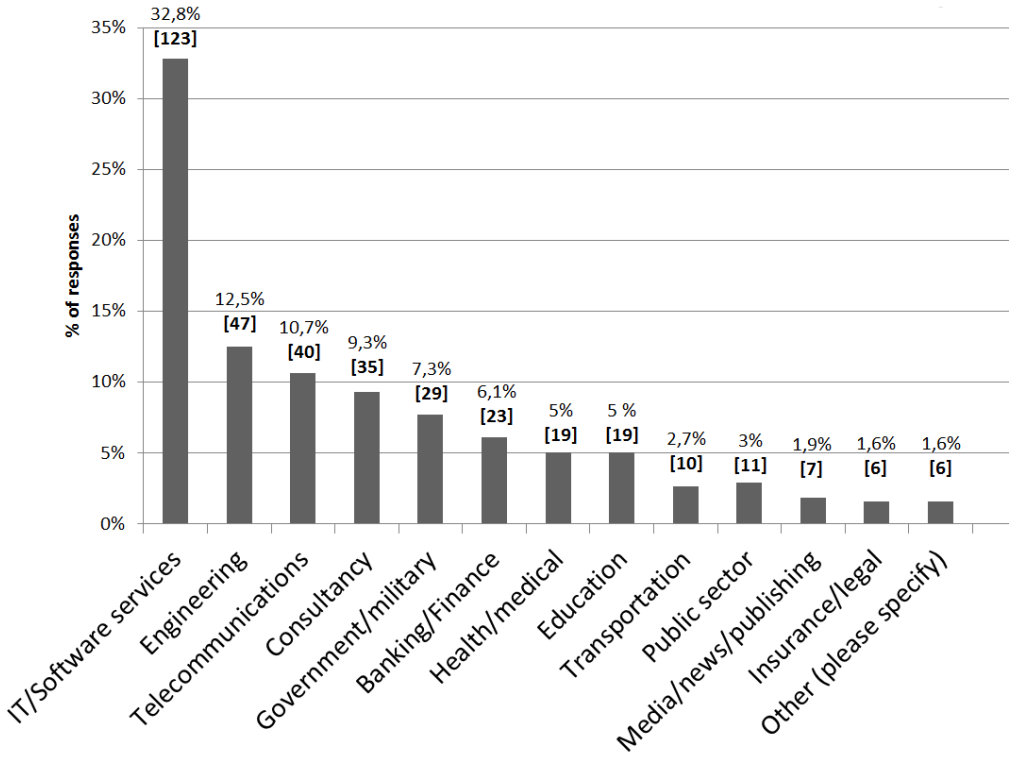

Figure 3: Types of business domains of respondents

quirements engineering the respondents are involved in (see Figure 7). Since this question also enabled multiple answers, the results are calculated based on the total number of answers. *Bespoke or Contract driven requirements engineering* received 44.2% of all the answers. *Market-driven requirements engineering* received 29.5%, while *Open source* only 5.1%. *Outsourced projects* appeared in 19.9% of the answers.

4.2. Defining obsolete requirements (RQ1)

Defining the term Obsolete Software Requirement (OSR) is central to the understanding of the phenomenon. The categories used in this question are inspired by the definitions of OSR found in literature (see Section 2), and are defined in the context of the current release. The answers from all respondents are depicted in Figure 8. Since the question enables multiple answers, the results are calculated for all the answers not respondents. The answer selected primarily by our respondents (29.9% of all answers, see Figure 8 de-

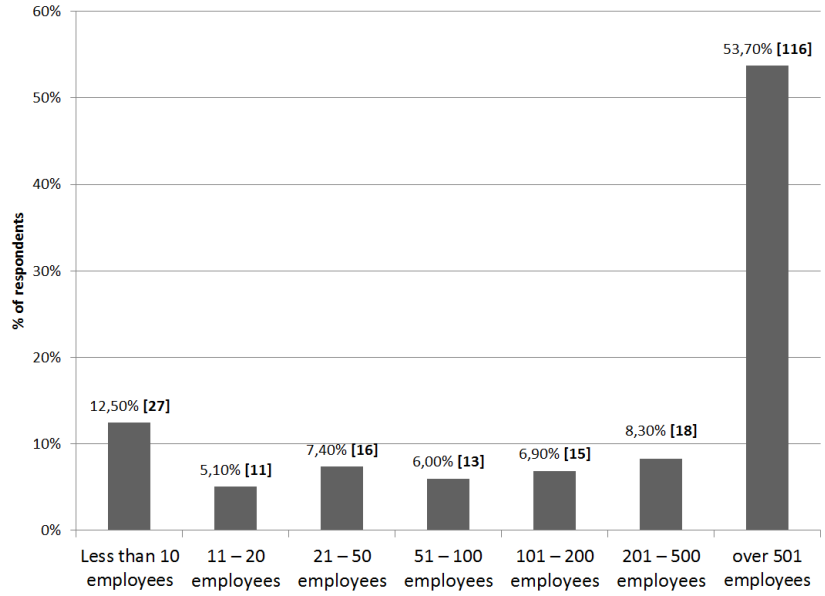

Figure 4: Size of respondents' organization

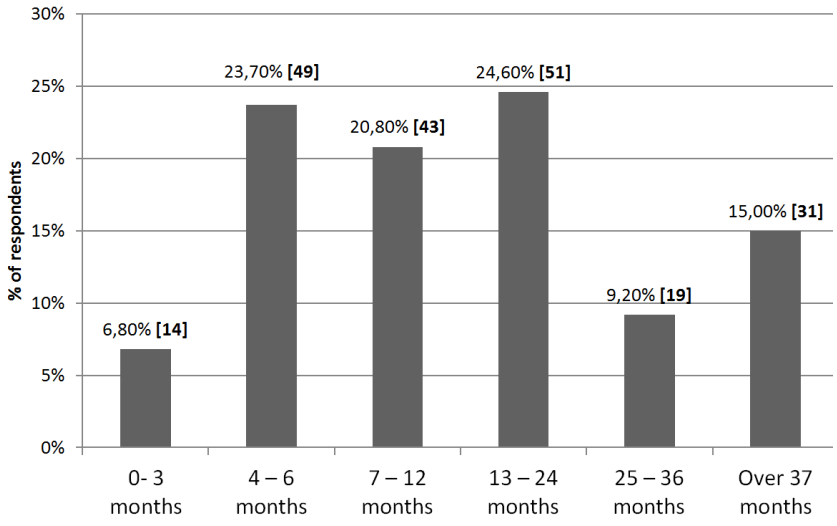

Figure 5: Average duration of typical projects from our respondents

fines OSR as "*no longer required for the current release for various reasons*". This result is in line with the definition of obsolete functionality provided by Zowghi et al. [35]. Further, 21% of the answers were given to the definition

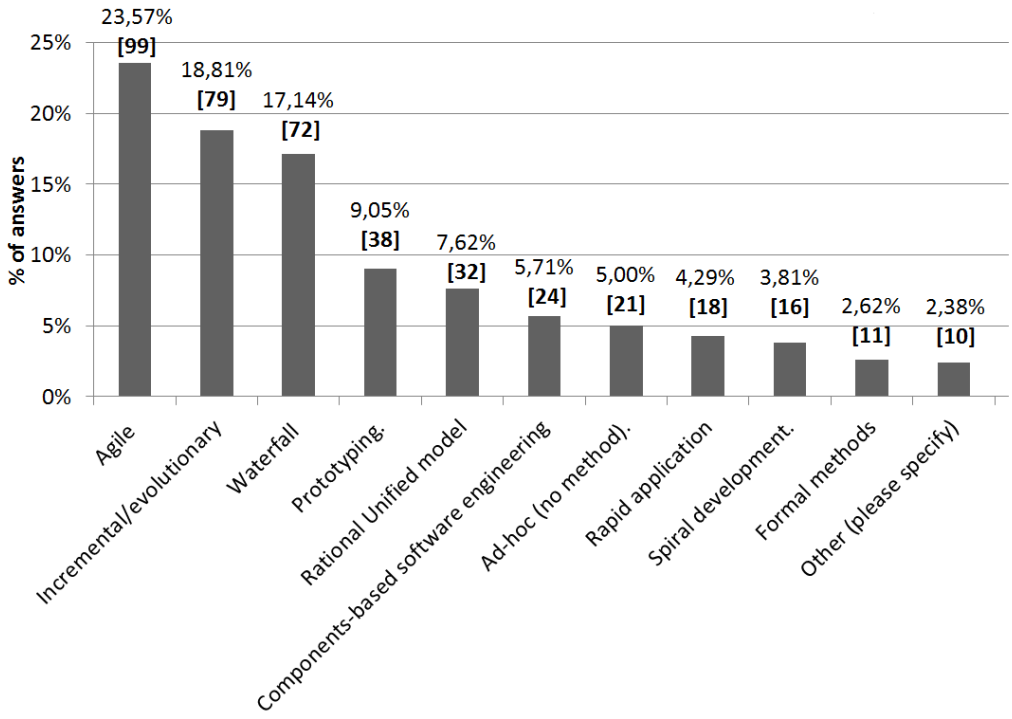

Figure 6: Development processes and methodologies

of an OSR as a requirement that: "*has no value for the potential users in the current release*". This category is similar to the definition of obsolete software applications provided by Merola [15], as applications taken off the market due to decrease in product popularity or other market factors.

A total of 33 responses (7.7%) were in the *Other* category. Out of these, 8 respondents ($\sim$ 25%) suggested that an OSR is not necessarily confined to the current release, but it also goes to future releases. Also, respondents stressed that an OSR is a requirement that has lost its business goal or value.

As a result, all points considered, the following definition of an OSR was formulated:

"An obsolete software requirement is a software requirement (implemented or not) that is no longer required for the current release or future releases, and it has no or little business value for the potential customers or users of a software product, for various reasons."

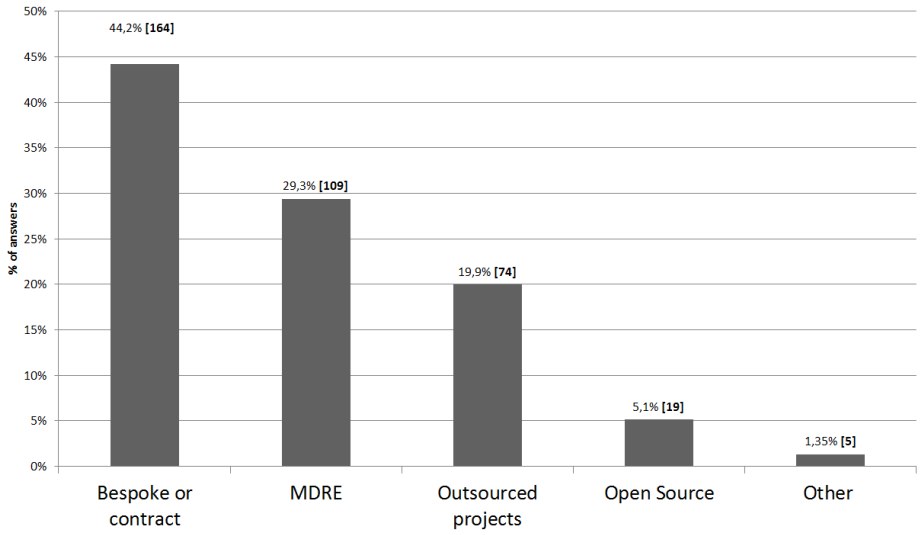

Figure 7: Types of Requirements Engineering

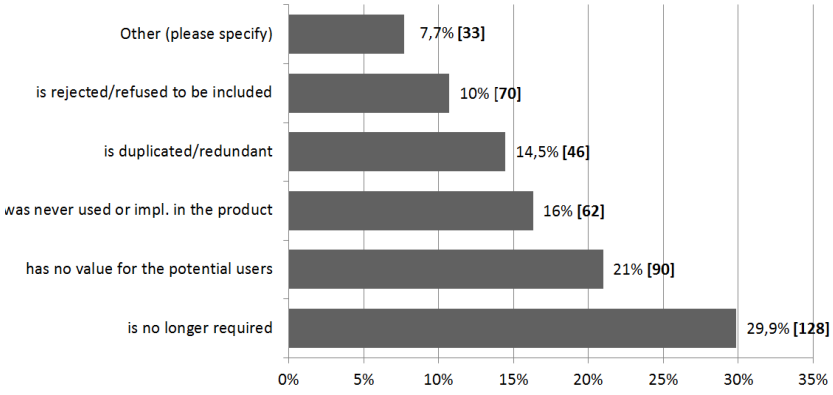

Figure 8: Respondents' definition of an OSR

As a next step we performed statistical analysis to investigate if there were relationships between the selected definition of OSRs and the respondents' roles, the size of organizations and the development methodologies used. Overall, the relationships turned out not to be statistically significant due to violations of the chi-square test assumptions (some answer alternatives had too few respondents, see Table A.2 in [54]). However, significant results could be observed (using chi-square test) between the top 5 methodologies

(see Figure 6) and the results for choice of OSR definition (p-value 0.011, see Table A.2a in [54]). Respondents that reported using a *Rational Unified Process* (RUP) methodology less frequently selected the definition of OSRs as (1) no longer required for the current release (31.3% of all answers comparing to over 50%) or (2) never used or implemented in the product (34.4% of all answers comparing to over 40%) than respondents that reported utilizing any of the remaining four methodologies. Moreover, the *RUP* respondents provided more answers in the *Other* category, and indicated that OSRs can be e.g. "a requirement that evolved in concept but not in documentation" or "an abstract requirement to showcase the technical capability to the end user". Finally, only three *RUP* respondents defined an obsolete software requirement as a requirement that is rejected to be included into the current release, while the respondents that selected the other top four methodologies selected this answer more frequently, around 20% of answers. This would seem to indicate that the consensus or perception as to what an obsolete software requirement is achieved with the help of the *RUP* methodology is more stable than achieved with the help of other methodologies.

Since the *RUP* methodology considers iterative development with continuous risk analysis as a core component of the method [56], the number of refused or rejected requirements should in this case be low, which confirms our results. Moreover the majority of the *RUP* respondents also reported to work with bespoke or contract-driven projects, where the number of changes after the contract is signed is limited and usually extensively negotiated. Thus it appears to be possible that the *RUP* respondents could avoid rejected or refused requirements and could manage to achieve more precise and stable agreements with their customers [56] which in turn could result in fewer OSRs.

Further analysis reveals that the definition of OSRs is not significantly related to the size of the companies, the length of the typical project, or the domain (p-values in both cases greater than 0.05). Domain and project length could be seen as qualifiers of OSR's, e.g. projects running over long periods could suffer increased requirements creep [8]. However, this would most probably not be visible in the definition of OSRs, rather in the impact of OSRs, which is investigated in the next section.

4.3. The potential impact of OSRs (RQ2)

When queried about the potential impact of OSRs on their product development efforts a total of 84.3% of all respondents considered OSR to be

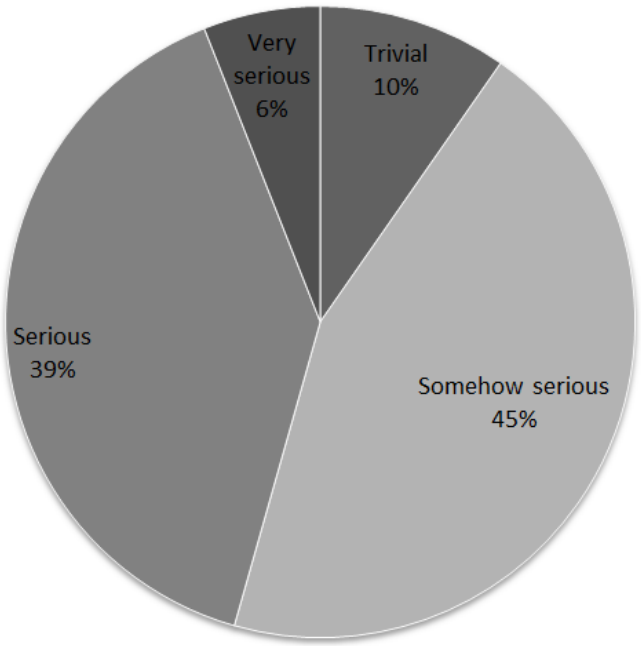

Figure 9: Impact of OSRs on industry practice

Serious or *Somehow serious* (see Figure 9). This indicates that among the majority of our respondents OSRs seems to have a substantial impact on the product development. Our result confirms previous experiences (see e.g. Murphy and Rooney [13], Stephen et al. [14] and Loesch and Ploederefer [18]). A total of 6% of the respondents considered OSRs as a *Very serious* issue, while 10% of the respondents (21 respondents) deemed OSR as a *Trivial* matter.

In an attempt to further decompose and test context variables (e.g. company size, respondents' roles and development methodologies) we performed chi-square tests (see Table A.1 in [54]) between the context variables and to what degree OSRs where considered having a substantial impact. The tests resulted in p-values greater than 0.05, which indicates that no statistically significant relationships between the analyzed factors could be seen. We can however ascertain that a clear majority of the respondents deemed the phenomenon of OSRs to be a relevant factor to be taken into consideration in development efforts.

Looking at the 21 (10%) respondents that considered OSRs to be *Trivial*, about 58% of them worked with requirements or in project management roles. This would seem to indicate that those respondents, contrary to software development roles, have less difficulty in managing OSRs, especially since the analysis of the answers to questionnaire question 9 (see [54] and Section 4.9) revealed that 10 respondents that considered OSRs to be *Trivial* also confirmed having a process of managing OSRs. Thus, it appears to be a logical conclusion that the negative influence of OSRs on the product development could be alleviated by designing and introducing an appropriate process of managing OSRs (more about the current processes discovered among out respondents can be found in Section 4.9).

Further analysis of the respondents who considered OSRs as *Trivial* indicated that more than 80% of them worked for large companies with $>$ 101 employees. Since large companies often use more complex process models [57], in contrary to small companies that might have budget constraints to hire highly quality professionals and their processes are typically informal and rather immature [58], we could assume that the issue of managing OSRs could have been already addressed in this case.

Further analysis of the *Trivial* group indicated that almost half of them (47.6%) worked in the *IT or computer and software services* domain, In the service domain, the main focus of requirements engineering is to identify the services that match system requirements [59]. In case of insufficient match of new requirements with the old system, product development may simply select a new more suitable service. This in turn might imply that the OSRs are discarded by replacing the old service with the new one. Further, the typical product lifetime for IT systems is usually shorter than for engineering-focused long-lead time products [60] (e.g. aerospace industry), which in turn could minimize the number of old and legacy requirements that have to be managed. The possible conclusion from our analysis is that OSRs are less of a critical issue in IT and service oriented domains.

Among the respondents that considered OSRs *Very serious* (13 respondents) the majority (53.8%) worked in large companies and used agile, adhoc or incremental methodologies (61.6%). This result seems to indicate that OSRs are also relevant for agile development and not reserved for only more traditional approaches like waterfall. Murphy et al. [13] stressed that the traditional (waterfall) requirements process seriously contributes to the creation of obsolete requirements by creating a "latency between the time the requirements are captured and implemented". This latency should be

less in agile projects (shorter iterations and deliveries). This might indicate that either the latency is present in agile projects as well, or that latency is not the primary determinant of OSRs. It should be observed that 69.2% of the respondents that considered OSRs as *Very serious* reported having no process of handling OSRs, which could indicate why OSRs were considered a *Very serious* problem.

The cumulative cross tabulation analysis of the respondents who considered OSRs *Somehow serious*, *Serious* or *Very serious* (total 196 respondents, 89%) confirmed the severe impact of OSRs on large market-driven and outsourced projects (see the results in Section 4.7.2). Moreover, 76.8% of those respondents reported that they had no process/method/tool for handling OSRs. Additionally 72.3% of respondents who considered OSRs *Somehow serious*, *Serious* or *Very serious* used manual ways to identify OSRs. It is also interesting to observe that these results, between the respondents who declared the following: *Agile software development* or *Incremental/evolutionary development* methodologies, and *Waterfall development*, only differ very slightly. Respondents using *Waterfall development* (and considered OSRs *Serious* or *Somehow serious* or *Very serious*) were somewhat more prone to dismiss the impact of OSRs compared to respondents using *Agile software development* or *Incremental/evolutionary* methodologies. This would seem to indicate that, because waterfall-like processes usually restrict late or unanticipated changes, and focusing on extensive documentation [61, 7, 62], the impact of OSRs in those processes could be minimized. However, it says nothing about to what extent the realized features were useful or usable for the customers. That is some waterfall projects may not have perceived OSRs to be a major issue for the project, but maybe for the product *per se*, which is a classic question of value perception in relation to perspective as described by Gorschek and Davis [63].

The type of requirements engineering context factor (see Figure 7) only minimally influenced the overall results for this questionnaire question. Respondents who reported to work with *Bespoke or contract driven requirements engineering* graded OSRs slightly less serious than respondents who reported to work with *MDRE*, which seems to indicate that OSRs are a problem in both contract driven (where renegotiation is possible [2]) and market-driven (where time to market is dominant [2]) projects. However, the difference could also indicate that there is a somewhat alleviating factor in contract-based development. That is, contract based development aims at delivering features and quality in relation to stated contract, thus getting

paid for a requirement even if it is out of date at delivery time. In a MDRE context however the product might fail to sell if the requirements are not fulfilled and the features out of date [2].

4.4. Requirements types and OSRs (RQ3)

The respondents were asked to choose what types of requirements were most likely to become obsolete (Likert scale, 1 = *Not likely*, and 5 = *Very likely*). The pre-defined "types" were derived from Harker et al. [36] and McGee and Greer [37].

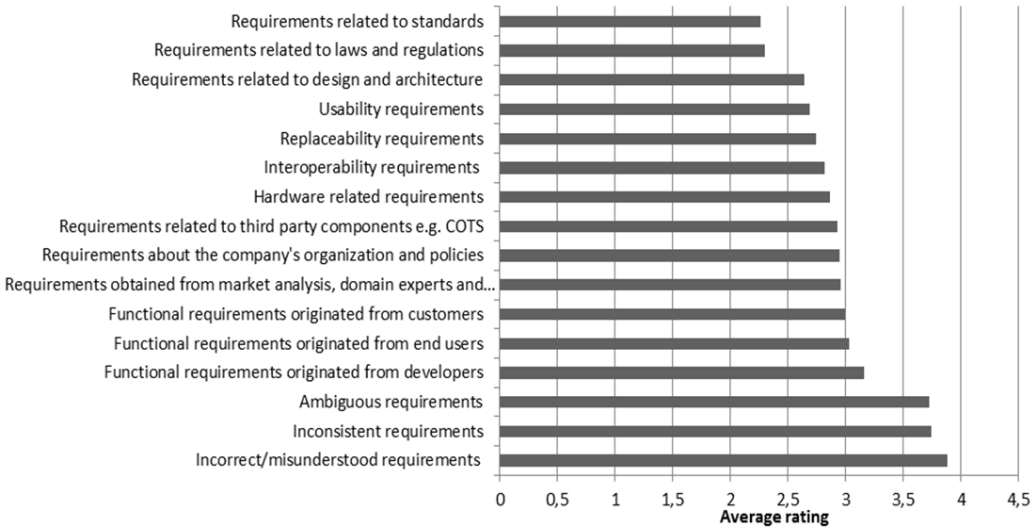

Figure 10: Types of OSRs likely to become obsolete

According to the results depicted in Figure 10, OSRs seem to belong to the categories of *Incorrect/misunderstood requirements* (mean 3.88), *Inconsistent requirements* (mean 3.74), and *Ambiguous requirements* (mean 3.72). From a becoming obsolete standpoint, the level and quality of specification should not matter *per se*. However, if quality of requirement's specification is seen as an indicator of a lack in investment (in analysis and specification) or a specific requirement several possible scenarios could emerge. For example, practitioners in industry might have a gut feeling that certain requirements will become OSRs, thus they are not worth the effort. Another possibility is that OSRs are harder (require more effort/knowledge) to specify than others, although, it could just as well indicate that most requirements are specified badly, thus also OSRs. To answer this, further investigation is needed. The

only thing we can say for sure is that requirements becoming obsolete seem to suffer from inadequacies in terms of correctness, consistency, and further seem to be ambiguously specified.

Interestingly, functional requirements, coming from domain experts, were considered less likely to become obsolete than requirements coming from customers, end users, and developers respectively. One explanation could be that domain experts possess the knowledge and experience of the domain, thus their requirements may be less likely to change [64]. On the other hand, since the customers are the main source of software requirements and the main source of economic benefits to the company, their requirements are crucial for the success of any software project [65], implying that this category needs to be kept up to date and thus more likely to become obsolete. Another possible explanation could be that customer requirements are not as well (unambiguously) specified as internal requirements [65, 22], resulting in a tendency of those requirements to become obsolete faster/more frequently.

Customer requirements becoming obsolete in excess to internal domain expert requirements is confirmed by Wnuk et al. [8], who reported that stakeholder priority dictates removal and postponement of the realization of requirements, and domain experts are often part of the prioritization of all requirements. On the other hand, Kabbedijk [66] reported that change request from external customers are more likely to be accepted than change requests from internal customers, which might imply that some customer requirements are handled as change requests instead of as requirements input to development projects. In both cases, the authors reported high volatility of requirements, which is in line with the study by Nurmuliand and Zowghi [35] who related obsolete requirements with requirements volatility.

According to our respondents, requirements related to standards, laws and regulations are the least likely to become obsolete, which seems logical, as the lifetime of legislation and standards is often long in comparison to customer requirements. Furthermore, the low average score for the *Requirements related to third party components e.g. COTS* (even lower than for the requirements related to the company's organization and policies) also seems to be logical, especially in relation to the results for RQ2 (see Section 4.3) where almost half of the respondents who considered OSRs to be *Trivial* worked with *IT or Computer and software services domain* (we assume after Bano and Ikram [59] that COTS are used in the software service domain). Moreover, the results for the respondents who worked with *Outsourced projects* (question 15 in [42]) are in accordance with the overall results. Also, the

differences between the respondents who worked with *Outsourced*, *MDRE* and *Bespoke or contract driven requirements engineering* projects in relation to the degree of obsolescence of COTS requirements are subtle. This may suggest that other aspects, not investigated in this study, could influence the results. Finally, albeit OSRs seems to be unrelated with one of the main challenges of COTS systems, i.e. the mismatch between the set of capabilities offered by COTS products and the system requirements [67], the nature of the COTS selection process, e.g. many possible systems to consider and possible frequent changes of the entire COTS solution when the current gets outdated, may help to avoid OSRs and their negative impact on the products.

Further analysis of the influence of the context factors indicates that the respondents' domains, company size, and methodologies have minimal impact on the results. Not surprising, more respondents who worked with projects running over longer timespans graded *Functional requirements originated from end users* as *Very likely* to become obsolete than respondents who worked with short projects (8.7% of respondents who worked with projects <1 year and 25.7% respondents who worked with projects > 1 year). One explanation could be that long projects, if deprived direct and frequent communication with their customers, and exposed to rapidly changing market situations, can face the risk of working on requirements that are obsolete from the users' point of view. This interpretation is to some extent supported by the results from RQ7 (see Table 4) where the respondents graded *MDRE* contexts (characterized by limited possibilities to directly contact the end users and continuously arriving requirements [2]) or *Outsourced projects* (where communication is often done across time zones and large distances [68]) as more affected by OSRs than bespoke contexts. Moreover, the success of *Market-driven projects* mainly depends on the market response to the proposed products [2], which if released with obsolete functionality may end up simply not bought by the customers. Thus, we believe that it is important to further investigate additional factors that could render *Functional requirements originated from end users* obsolete.

4.5. Methods to identify OSRs (RQ4)

More than 50% of the answers pointed out that manual ways of discovering OSRs are being the primary method (see Figure 11). At the same time, the context factors such as the different methodologies, types of RE, length of the projects, roles of respondents or the domain that respondents worked in turned out not to significantly affect the top answer for this question. A

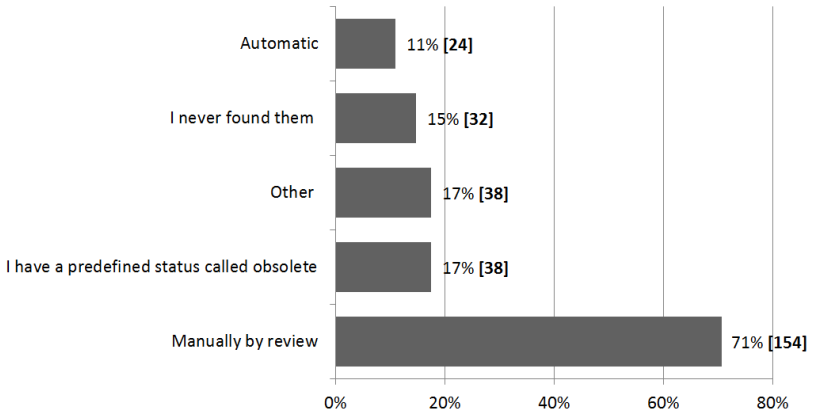

Figure 11: Methods used to identify OSRs

total of 13.29% of all answers indicated having a predefined status in their requirements handling systems for OSRs. Furthermore, 11.19% of all answers (32 answers) were given to the category *I never found them or I never thought of finding them*. Finally, less than 10% of all answers (24 answers) indicated the existence of any sort of automation in terms of identification of OSRs.

The answers from respondents who indicated using automated ways of discovering OSRs provided some names for the automated techniques, e.g. "customized system based on JIRA that takes OSRs into account by using special view filters", "traceability using DOORs to analyze for orphan and to track and status obsolete requirements", or "a tool called Aligned Elements to detect any inconsistencies including not implemented requirements". This would indicate that some tool support is present, however tool efficiency and effectiveness was not part of the study.

Further analysis indicated that the majority of respondents using tools of some sort worked with companies > 501 employees (62%). This seems reasonable as large companies usually have more money for tool support [58], and can even request especially tailored software from the requirements management tool vendors. The fact that automated methods to identify OSRs are rare among the smaller companies calls for further research for lightweight and inexpensive methods of OSRs identification that can more easily be adapted in those companies.

On the other hand, more than half (15) of the respondents from the automated group also indicated that they identify OSRs manually. One ex-

planation could be that automated methods are used together with manual methods, e.g. after the respondents actually manually mark requirements as obsolete or perform other preliminary analysis that enables automated sorting. Searching, tagging or filtering capabilities in their requirements management tools are most likely dominant and seen as *automated* in relation to OSRs, but this task is done in an ad-hoc manner and not integrated with their requirements management process. Thus the level of "automation" needs further investigation.

The reasonably high number of answers given to the category *I never found them or I never thought of finding them* is intriguing and needs further investigation. A total of 93.8% (30) of the respondents from this group also indicated having no process of managing OSRs, which seems logical as the inability to find OSRs could be related to the lack of processes of managing OSRs. Further, the majority of these respondents (that indicated never finding OSRs) worked with projects shorter than 12 months, and one fourth of them indicated having an ad-hoc process of managing requirements. The relatively short project times were not an indication of OSRs nor being an issue as >80% of these same respondents indicated OSRs as being a *Serious* or *Very serious* issue. The absence of a defined and repeatable process might be a better indicator for not identifying OSRs in this case. In addition, in relation to the respondents who never thought of finding OSRs, waterfall was represented in more than 11% of the cases, while only about 6% worked in an agile manner.

Using statistical analysis, organizational size and development methodology was not a statistically significant factor in terms of how OSRs were identified or found (see Table A.5 in [54] for details). However, a statistically significant relationship was identified in relation to the top five methodologies used by our respondents, and ways how OSRs were identified (chi-square test $p<0.004$, see Table A.5a in [54] for details). This result could be explained by the following: (1) respondents who worked with waterfall methodology more often admitted to never find OSRs (11%) than respondents who worked with agile methodologies (3.8%), (2) more respondent who worked with *RUP* methodology (34%) selected the option *I have a predefined status called obsolete* than respondents who worked with agile methodology (10%). Looking further, we could also see that the majority of the respondents who worked with *RUP* or *Prototyping* methodologies also worked with companies with >201 employees. This would seem to indicate that for within the two mentioned methodologies it is possible to implement tool support for identifica-

tion of OSRs.

When it comes to analyzing the influence of the types of requirements engineering used, the results showed that the respondents who work with *Bespoke or contract driven requirements engineering* didn't use predefined categories for OSRs, that is, it was not part of their standard procedure to sort our OSRs, which could be logical as the majority of the respondent who admitted to never finding OSRs worked with bespoke or contract-driven projects (see the analysis above). Finally, automatic methods for findings OSRs turned out to be unpopular among the respondents who worked with open sources projects (only one respondent mentioned it).

For the context factor of project length, the results indicate that longer projects have more automated ways of identifying OSRs (the difference is about 5% of the votes) than shorter projects. This seems reasonable as longer projects usually invest more into project infrastructure and project management tools and processes. However, a large part of the longer projects respondents also indicated manual methods of identifying OSRs (about 60% of all answers for projects >1year). In comparison, subjects working typically in shorter projects used more tool supported automated methods (about 52% of all answers for projects <1 year). Thus the respondents working in longer projects did see the point of, and did try to, identify OSRs to a larger extent than the ones working in shorter duration projects, although manual methods dominated.

The analysis of the influence of the respondents' roles on the results revealed only minimal differences. Interestingly however the roles of project and product managers where the only respondents who had no answers in the I never found them category, indicating that they always find OSRs. Further, the management roles had the highest score for manual identification of OSRs. This result might indicate that management is to some extent more aware of the need for finding OSRs as they may severely impede the project efforts, however tool support is often lacking.

4.6. Handling of identified obsolete software requirements (RQ5)

More than 60% of the answers indicated that the respondents (results for multiple answer questions are calculated based on all the answers) kept the OSRs but assign them a status called "obsolete" (see Figure 12). This might indicate that OSRs are a useful source of the information about the history of the software product for both requirements analysts and software development roles. Moreover, 21.85% of all answers (66) suggested moving

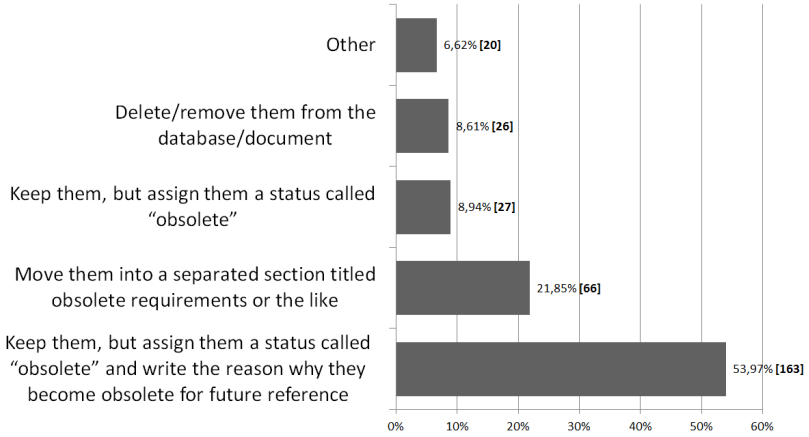

Figure 12: Methods used to manage identified OSRs

OSRs into a separated section in requirements documents. These views were the most popular among the respondents regardless of their role, methodology, domain, size, project length and context reported. The interpretation that could be derived from this result is that the most suitable way to manage identified OSRs is to classify them as obsolete, supplying rationale, and move them into a separated section/document/SRS. However, maintaining traceability links between OSRs and other requirements could prove work intensive, especially if end-to-end traceability is required [57]. Wnuk et al. [44] discusses scalable methods for managing requirements information where effective grouping of requirements (e.g. placing semantically similar requirements in the same module) could enable more efficient maintenance of large structures of requirements (although OSRs where not mentioned specifically).

Most of the answers in the other category (∼6%, 20 answers) suggested either removing OSRs, or keeping them, but moving to a separated section or module in the database. Only ∼9% of answers (26) suggested deleting the OSRs from the requirements database or document, which suggests that most respondents think OSRs should be stored for reference and traceability reasons. Keeping OSRs appears to be inconsistent with recommended practice for reducing the complexity of large and very large projects [44, 69], and handling information overload as highlighted by Regnell et al. [44]. The desired behavior in large and very large project would seem to indicate the removal of unnecessary requirements to decrease the complexity of the re-

quirements structure and traceability links. One possible avenue for further investigation is to evaluate what value keeping OSRs provides.

Out of the group who opted for OSRs deletion upon identification, the majority of the answers came from respondents who worked with large companies (>501 employees, 77%) and long projects (>12 months, 53.9%). Moreover, a majority of these respondents considered OSRs to be *Serious* or *Somehow serious* (see Section 4.3). On the contrary, respondents that worked in smaller companies opted to keep OSRs.

Analysis revealed a lack of statistical significant relationships between the answers for this question (see Figure 12) and the respondents' roles, domains, organizational size, as well as methodologies used (see Table A.6 in [54]). However, some indications could be observed. Respondents working in the engineering domain seemed to prefer the deletion of OSRs compared to respondents from other domains. One possible explanation could be that since the projects in the engineering domain are highly regulated, and often require end-to-end traceability [57], keeping OSRs in the scope could clutter the focus threatening to impede requirements and project management activities.

Type of requirements engineering factor turned out to have a minimal impact, however, one observation that is worth mentioning is that more answers were given to the option of removing OSRs among the respondents who worked with *Bespoke or contract driven requirements engineering* (12.3%) than respondents who worked in *MDRE* (9.2% of answers). This appears to be logical as in bespoke projects obsolete requirements could be discarded after the contract is fulfilled, while for market-driven projects they could be kept and later used during requirements consolidation task, where new incoming requirements could be examined against already implemented or analyzed requirements which include OSRs [70] .

4.7. Context factors and obsolete software requirements (RQ6 and RQ7)

4.7.1. Obsolete software requirements and project size

The respondents were asked to indicate to what extent the phenomenon of OSRs would potentially (negatively) impact a project, and if project size had anything to do with the likelihood of negative impact. The respondents used a Likert scale from 1 (*Not likely* impact) to 5 (a *Very likely* impact). The results are presented in Tables 3 and 4 below. The size classification is graded in relation to amount of requirements and interdependencies, inspired by Regnell et al. [44].

Table 3: OSRs effect on project size (215/219 respondents)

	(1) Not likely	**(2) Some-what likely**	**(3) Likely**	**(4) More than likely**	**(5) Very likely**	**Rating Aver-age**
Small-scale (∼10 of req.)	35.3% (76)	**35.8% (77)**	13.5% (29)	7.0% (15)	8.4% (18)	2.17
Medium-scale (∼100 of req.)	9% (19)	31.6% (67)	**41.5% (88)**	16.0% (34)	1.9% (4)	2.70
Large-scale (∼1000 of req.)	3.8% (8)	17.1% (36)	31.3% (66)	**32.7% (69)**	15.2% (32)	3.38
Very large-scale (>10000 of req.)	8.1% (17)	12.8% (27)	16.6% (35)	23.7% (50)	**38.9% (82)**	3.73

Column 7 in Table 3 presents the average rating for each project size, where we can see a connection between OSRs effect and project size, that is the larger the project the more likely a negative effect of OSRs as indicated by the respondents. Looking at Table 3, for *Small-scale requirements* projects most respondents deemed OSR impact as *Not likely* (35.3%) or *Somewhat likely* (35.8%). However, moving up just one category to *Medium-scale requirements* projects (hundreds of requirements) the respondents indicated the impact as being *Likely* (41.5%). The trend continues with *More than likely* (32.7) for *Large-scale requirements* projects, and *Very likely* for *Very large-scale requirements* projects (38.9%). The results confirm the viewpoint of Herald et al. [20], who listed OSRs as one of the risks in large integrated systems.

One interesting observation is that the results could be seen as potentially contradictory with the results from questionnaire question 2 (see Section 4.3), that is respondents who worked in larger companies (over 100 employees) graded the overall impact of OSRs slightly less than respondents from smaller companies. This could suggest that larger companies do not nessecaraly have larger (number of requirements) projects, or, that there are other factors that influence the impact of OSRs.

Table 4: How likely OSRs affect various project types (215/219 respondents)

	(1) Not likely	**(2) Somewhat likely**	**(3) Likely**	**(4) More than likely**	**(5) Very likely**	**Rating Average**
Bespoke projects	14.4% (31)	**32.1% (69)**	26% (56)	16.3% (35)	11.2% (24)	2.78
Market-driven projects	6.5% (14)	20% (43)	**35.8% (77)**	23.3% (50)	14.4% (31)	3.19
Outsourced projects	2.3% (5)	16.4% (35)	**35.7% (76)**	27.2% (58)	18.3% (39)	3.43

When it comes to the influence of methodology used by our respondents, we report that the respondents who used *Agile software development* methodology primarily graded OSRs as only *Likely* to affect *Large-scale requirements* projects, while respondents who used *Waterfall* methodology primarily graded the impact of OSRs as *More likely*. Interestingly, this result seems to contradict the results for RQ2 (see Section 4.3), where the majority of respondents who considered OSRs *Very serious* worked in large companies and used agile or incremental methodologies. This might indicate that the size of the project is not more dominant than the size of the company, and the methodology used. This requires further investigation.

The respondents who worked with bespoke or contract driven requirements engineering primarily graded the effect of OSRs on *Large-scale requirements* projects as *Likely*. On the contrary, the respondents who worked with *Market-driven projects* primarily graded the impact of OSRs on *Large-scale requirements* projects as *Very Likely*. This result confirms the results for RQ2 (see Section 4.3) where OSRs were also graded less serious by respondents who worked in bespoke contexts. Finally, for the *Very large-scale requirements* projects our respondents primarily graded the impact of OSRs as *Very likely* regardless of context factors.

4.7.2. Obsolete software requirements and project types

The respondents were also asked to rate how likely it was that OSRs affected various project types (on a scale from 1 to 5, where 1 is *Not likely*, and 5 is *Very likely*). The results for the average rating (column 7 in Table 4) indicate that *Outsourced projects* are the most likely to be affected by OSRs (average rating 3.43). One possible explanation for this result could be the inherited difficulties of frequent and direct communication with customers and end users in *Outsourced projects*. Moreover, as communication in *Outsourced projects* often needs to be done across time zones and large distances [68, 71], the risk of requirements misunderstanding increases, and as we have seen (see Section 4.4), inadequately specified requirements run a higher risk of becoming OSRs.

The high average rating for the *Market-driven projects* (average scope 3.19) can be explained by the inherited characteristics of the MDRE context where it is crucial to follow the market and customer needs and the direct communication with the customer may be limited [2]. This in turn can result in frequent scope changes [8] that may render requirements obsolete. Finally, it is worth mentioning that the gap between the *Market-driven projects* and *Bespoke projects* (average score 2.78) is wider than between *Outsourced* (average scope 3.43) and *Market-driven projects* (average score 3.19). One possible explanation could be that both *Market-driven projects* and *Outsourced projects* suffer similar difficulties in directly interacting with the end users or customers [2, 68] and thus the risk of requirements becoming obsolete could be higher.

The results for all the categories and scales are presented in columns 2 to 6 in Table 4. Our respondents primarily graded the impact of OSRs on *Market-driven projects* and *Outsourced projects* as *Likely* and only *Somehow likely* for *Bespoke projects*. Interestingly, the answer *Very likely* did not receive top scores for any of the three types of projects. This would seem to indicate that the "project type" factor is less dominant in relation to OSRs than the "size" of the project discussed earlier in this section.

Since the statistical analysis between the results from the question and the context variables revealed no significant relationships, we performed descriptive analysis of the results. The respondents who indicated having a managerial role (32.7%) primarily graded the impact of OSRs on the *Market-driven projects* as *More than likely*, while the requirements analysts primarily graded this impact as only *Likely*. Similarly to this result are the results for

RQ2 (see Section 4.3), where the managers primarily considered that OSRs are *Serious* while the answer from requirements analysts was predominantly *Somehow serious*. The comparison is however not straight forward as in case of RQ2 respondents were grading all types of requirements projects, not only *Bespoke projects*. Finally, the opinions of software development and management roles are inline when grading the impact of OSRs on bespoke projects (the majority of the respondents from both roles graded the impact as *Somehow likely*).

In relation to project duration, interestingly, respondents who worked with smaller companies (<200 employees) more often graded the effect of OSRs on *Bespoke projects, Market-driven projects* or *Outsourced projects* as *Likely* or even *Very likely*. The majority of the respondents who worked for companies with > 201 employees selected the *Somehow likely* answer for the *Bespoke projects* and *Market-driven projects*. This result confirms the previous analysis (see Section 4.7.1) by indicating that size is not the only factor that impacts the seriousness of OSRs. It can also be speculated that the phenomenon of OSRs might be more clear in smaller organizations where less specialization makes outdated requirements more of a "everybody's concern", while in lager organizations, with high specialization, the view of "not my job" might play a factor [57].

4.8. Where in the requirements life cycle should OSRs be handled (RQ7)

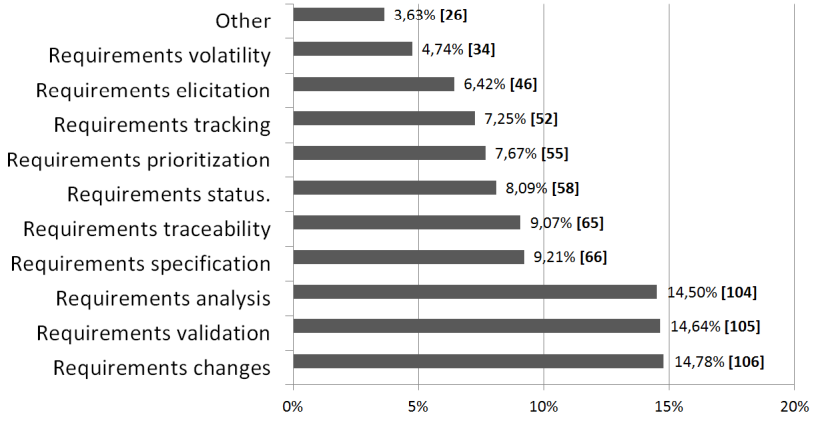

Figure 13: Requirements lifecycle stages for addressing OSRs

The results for this question are presented in Figure 13 below as percentages of the total number of answers (717) since the question enabled

multiple answers. The list of phases or processes in the requirements engineering lifecycle was inspired by Nurmuliani and Zowghi [35] (called phases from now on). According to our respondents OSRs should first of all be handled during *Requirements analysis, Requirements validation* and *Requirements changes* phases (each with about 14% of the answers). This result is to some extend in line with the study by Murphy and Rooney [13], SWEBOOK [34] and Zowghi et al. [35] who report that change leads to volatility, and volatility in its turn leads to obsolescence. However, the results from our survey, where less than 5% of the answers indicate that OSRs should be manged as a part of the requirements volatility handling seems to support a distinction between volatility and the phenomenon of OSRs as such. That is, volatility may be related to OSRs, however, it needs to be handled continuously during analysis and validation, as a part of change management in general.

The high numbers of answers given to *Requirements analysis* (14.5%) and *Requirements specification* (9.2%) phases confirm the suggestions made by Savolainen et al. [17] to manage OSRs in the requirements analysis phases. The low score of *Requirements elicitation* phase answer (6.42% of all answers) contradicts with the viewpoint of Merola [15] who suggested managing obsolete software by continuous and timely market tracking and market trends changes identification. This might seem to indicate that our respondents have difficulties in understanding how OSRs could be managed e.g. by finding and dismissing OSRs faster due to continuous elicitation depending on the accepted definition of OSRs.

Respondents working with *Agile software development* methodologies preferred to handle OSRs as a part of the *Requirements changes* phase, while respondents working in a *Waterfall* manner preferred the *Requirements validation* phase. This seems logical, as a part of agile methodology is to embrace change [4], while waterfall philosophy sees it as to be "handled" more formally as steps in development (focusing on the specification and validation phases) [23].

Type of requirements engineering context (see Figure 7) did not seem to matter, i.e. analysis, validation, and changes phases seemed to be dominant for *MDRE* and *Bespoke or contract driven requirements engineering* alike. However, looking at company size and project duration, respondents from larger companies with longer projects focused on handling OSRs in specific phases, i.e. analysis and validation. This result seems reasonable as large projects usually require more extensive requirements analysis due to e.g.

larger number of stakeholders involved and possible higher complexity of the system to be developed [44, 57, 69].

4.9. Existing processes and practices regarding managing OSRs (RQ5)

When queried about the existing processes and practices regarding managing OSRs, 73.6% of all respondents (159) indicated that their requirements engineering process does not take OSRs into consideration. This result can be interpreted as a clear evidence of lack of methods regarding OSRs in industry, and confirms the need for developing methods for managing OSRs. At the same time, some processes for managing OSRs exist (as indicated by 26.4% (57) of our respondents). Below a list of processes/methods used by our respondents:

- Reviews of requirements and requirements specifications (19 respondents)
- Using tools and marking requirements as "obsolete" (6 respondents)
- Requirements traceability (6 respondents)
- Discussing and prioritizing requirements with customers in agile context (4 respondents)
- Mark obsolete requirements as "obsolete" (4 respondents) - these respondents did not indicate if using the tool or not.
- During the requirements management process by identifying OSRs (3 respondents)
- Moving OSRs into a separated section in the SRS (3 respondents)
- Through change management process (2 respondents)
- During the requirements analysis process (1 respondent)
- Having a proprietary process (1 respondent)

The identified "categories" of processes/methods above overlap with previous results from the survey. For example, the process of managing OSRs by requirements reviews overlap with the most popular way to identify OSRs (see Figure 11 in Section 4.5), as indicated by our respondents. This would

seem to indicate that manually reviewing requirements is dominant. Whether or not this is sufficient is another question which needs to be investigated further. Moreover, the results confirm what was reported in Section 4.5 that automated methods for identification and management of OSRs are rare. Therefore, further research of scalable automatic methods for identification and management of OSRs is needed.

Some respondents provided names of descriptions of processes/methods used for managing OSRs. Among the reported are:

- *Projective analysis through modeling* - considered as a promising approach to study the complexity pertaining to systems of systems [72], but requires a skilled "process modeler" to seamlessly use the modeling paradigm. If and how the method could be applied for smaller projects, and particularly for identification and management of OSRs remains an open question. Also, the technique is used during the requirements analysis phase which has been considered a good phase of management of OSRs by our respondents (see Figure 13).

- *Hierarchical requirements' tables* - specifying requirements on different abstraction levels is one of the fundamental techniques of requirements engineering that helps various stakeholders to understand requirements better [22]. Considered as one of the requirements specification techniques, this technique could be promising according to our respondents (see Figure 13). However, this method can be used to control OSRs to a certain degree as overview of requirements mass can be achieved to some extent through abstraction [65]. However, given huge amounts of requirements scalability of the method could be a problem.

- *Project governance* - considered as supporting project control activities considering the environment in which project management is performed [73]. By having a wider time scope than ordinary project management, project governance could, according to our interpretation, be supportive in the continuous task of identification and management of OSRs.

- *Requirements tracking with risk management* - although we consider tracking and risk management [22] as separated activities, combining them for the purpose of managing OSRs is an interesting alternative potential. In particular, the role of risk management in identification

and management of OSRs should be further investigated, as the software risk management literature seem to not mention OSRs as one of the software risks [74].

- *Requirements-based test plans* - aligning requirements with verification, although challenging, could be considered critical in assuring that the developed software fulfills customers' needs. Creating test plans based on requirements that are up-to-date and properly reflect changing customer needs is appreciated as good practice in software projects [75]. We are however unsure to what degree the practice of writing test plans based on requirement could help in identification and management of OSRs. The fact that test plans are based on requirements is to us independent of the fact that these requirements may simply be obsolete.

- *Commenting obsolete code and updating requirements documents accordingly* - this technique of managing OSRs could be considered promising and should help to keep the requirements aligned with the newest version of the code. However, the technique seems to only consider implemented requirements that could be directly traced to code level. Considering the fact that many requirements (especially quality requirements) are cross-cutting and require implementation in several places [22] in the source code, in our opinion, it could be challenging to correctly map changes in the code to changes in requirements. This could be part of a solution, however lacks the possibility to identify and remove OSRs prior to implementation.

- *Using requirements validation techniques to identify if requirements are no longer needed* - validating requirements is fundamental for assuring that the customer needs were properly and correctly understood by the development organization [22]. In our opinion, this technique should be used together with customers who can confirm if the requirements are relevant. Our respondents also would like OSRs to be managed during requirements validation phase (see Figure 13). However, if requirements reviews are conducted in isolation from "customers", by e.g. requirements analysts, they could have difficulties in identifying which requirements are, or are about to become obsolete. This is further aggravated if the development organization operates in a MDRE context.

Looking at the context factors of organizational size, development methodology, and respondent role, although no statistical significant correlations could be observed, some interesting points warrant mention. Respondents from smaller companies (<50 employees) to a larger degree had explicit practices for handling OSRs as compared to respondents from larger companies. This seems reasonable when looking at the methods for managing OSRs provided, where manual review methods were most frequent. Quispire [58] mentioned that processes used in small software enterprises are often manually based and less automated.

Respondents who worked with *MDRE* projects (see Figure 7) more often reported having processes that take OSRs into consideration (34.3%), than respondents who worked with *Bespoke or contract driven requirements engineering* (26.5%) or *Outsourced projects* (15.8%) respectively (almost significant results with a p-value of 0.059, see Table A.8a in [54]). One possible explanation for this could be high and constant requirements influx in MDRE contexts [2, 44], combined with frequent changes to requirements dictated by rapidly changing market situation. This in turn is resulting in more requirements becoming obsolete, forcing the use of methods to manage OSRs.

Further statistical tests (reported in Table A.8 in [54]) indicated a statistical significance between the roles of respondents and the existence of processes to manage OSRs ($p = 0.0012$). There was also a moderate association (Cramer's $V = 0.345$) between the respondents' roles and the existence of requirements engineering processes that take OSRs into account. From the cross-tabulation table between the respondents' roles and the existence of OSRs handling process (Table A.9 in [54]) we can see that the respondents who worked in management positions (project and product managers) were more likely to utilize OSRs handling method, as compared to respondents who worked in software development roles (e.g. developers).

Further, the presence of a method/process that considers OSRs seems to decrease the negative impact of OSRs among our respondents, as 50% of the respondents who deemed OSRs *Trivial* confirmed to have a process of managing OSRs (see Section 4.3). Moreover, as requirements engineers as well as product and project managers usually work more with requirements engineering related tasks than software development roles, it appears to be logical that more methods of managing OSRs are reported among the management roles.

4.10. Summary of results

The results from the study could be summarized in the following points:

- Our respondents defined an OSR (RQ1) as: "a software requirement (implemented or not) that is no longer required for the current release or future releases, and it has no or little business value for the potential customers or users of a software product, for various reasons." This definition seems to be homogeneous among our respondents (with a small exception for the respondents who used *RUP* methodologies).

- OSRs constitute a significant challenge for companies developing software intensive products, with the possible exception of companies involved in the service domain. The phenomenon of OSRs is considered serious by 84.3% of our respondents (RQ2). At the same time 73.6% of our respondents reported having no process of handling obsolete software requirements (RQ5).

- Requirements related to standards and laws are the least likely to become obsolete, while inconsistent and ambiguous requirements are the most likely to become obsolete (RQ3). Moreover, requirements originating from domain experts were less likely to become obsolete than requirements originating from customers or (internal) developers.

- OSRs identification is predominantly a manual activity, and less than 10% of the respondents reported having any automated functionality - which suggests research opportunities in creating automated methods for OSR identification and management (RQ4).

- The identified OSRs should, according to more than 60% of the survey answers, be kept in the requirements document or the database, but tagged as obsolete. Deleting OSRs is not a desired behavior (RQ5). Most respondents opted for keeping the OSRs for purposes of reference and traceability, which seems to indicate that the identification of OSRs is not the only action, but a wish to potentially use the OSRs to minimize double work (e.g. specifying new requirements that are same or similar as already identified OSRs). This is especially relevant in the MDRE context where "good ideas" can resurface as proposed by e.g. internal developers.

- Although there exist methods and tool support for the identification and handling of OSRs, a clear majority of the respondents indicated no use of methods or tools to support them, rather, ad-hoc and manual process seemed to dominate (RQ5). Moreover, even when the identification of OSRs was deemed central (e.g. for respondents working in longer duration projects), only some tool support and automation was present (for mostly bespoke projects), but even here manual processes and routines dominated (see Section 4.5).

- Project managers and product managers indicate that they always find OSRs in their work (see Section 4.5), even if many of the respondents don't actively look for them.

- OSRs are more likely to affect *Large-scale requirements* and *Very large-scale requirements* projects (RQ6). Larger projects (hundreds of requirements) tend to have larger issues related to the presence of OSRs, and there seems to be a correlation between impact severity and project size (amount of requirements). OSRs seem to have a somewhat larger impact on projects in a MDRE context as compared to bespoke or contract driven development (see Section 4.7.2). However, for very-large projects (over 10 000 requirements) all respondents, independent of context factors, agree that the potential impact of OSRs was substantial.

- According to the respondents, OSRs should first of all be handled during the *Requirements analysis* and *Requirements validation* phases (RQ7). At the same time, less than 5% of the answers indicate that OSRs should be managed as a part of requirements volatility handling which supports the distinction between volatility and the phenomenon of OSRs as such. Finally, our respondents suggested that *Requirements elicitation* is not the best phase to manage OSRs.

- Latency may not be the main determinant of OSRs becoming a problem, rather the results point to the lack of methods/routines for actively handling OSRs as a central determinant. This would imply that claimed low latency development models, like agile, has and can have problems with OSRs.

5. Conclusions and Further Work

Although the phenomenon of obsolete software requirements and its negative effects on project timelines and the outcomes have been reported in a number of publications [9, 13, 14, 15, 7], little empirical evidence exists that explicitly and exhaustively investigates the phenomenon of OSRs.

In this paper, we report results from a survey conducted among 219 respondents from 45 countries exploring the phenomenon of OSRs by: (1) eliciting a definition of OSRs as seen by practitioners in industry,, (2) explore ways to identify and manage OSRs in requirements documents and databases, (3) investigating the potential impact of OSRs, (4) explore effects of project context factors on OSRs, and (5) define what types of requirements are most likely to become obsolete.

Our results clearly indicate that OSRs are a significant challenge for companies developing software systems - OSRs were considered serious by 84.3% of our respondents. Moreover, a clear majority of the respondents indicated no use of methods or tools to support identification and handling OSRs, and only 10% of our respondents reported having automated support. This indicates that there is a need for developing (automated) methods or tools to support practitioners in the identification and management of OSRs. These proposed methods need to have effective mechanisms for storing requirements tagged as OSRs, enabling the use of the OSRs mass as decision support for future requirements and their analysis. This could potentially enable automated regression analysis of active requirements, continuously identifying candidates for OSRs, and flagging them for analysis.

Although manually managing OSRs is currently the dominant procedure, which could be sufficient in small projects, research effort should be directed towards developing scalable methods for managing OSRs - that scale to reality which is often characterized by large amounts of requirements and a continuous substantial influx of new requirements. The reality facing many product development organizations developing software intensive systems today is that OSRs are a problem, and as the amount and complexity of software increases so will the impact of OSRs.

References

[1] M. DeBellis, C. Haapala, User-centric software engineering, IEEE Expert 10 (1) (1995) 34 –41. doi:10.1109/64.391959.

[2] B. Regnell, S. Brinkkemper, Market-driven requirements engineering for software products, in: A. Aurum, C. Wohlin (Eds.), Engineering and Managing Software Requirements, Springer Berlin Heidelberg, 2005, pp. 287–308.

[3] T. Gorschek, S. Fricker, K. Palm, S. Kunsman, A lightweight innovation process for software-intensive product development, Software, IEEE 27 (1) (2010) 37 –45. doi:10.1109/MS.2009.164.

[4] B. Ramesh, L. Cao, R. Baskerville, Agile requirements engineering practices and challenges: an empirical study, Inf. Syst. J. 20 (5) (2010) 449–480.

[5] T. Gorschek, M. Svahnberg, A. Borg, A. Loconsole, J. Börstler, K. Sandahl, M. Eriksson, A controlled empirical evaluation of a requirements abstraction model, Inf. Softw. Technol. 49 (2007) 790–805. doi:http://dx.doi.org/10.1016/j.infsof.2006.09.003.
URL http://dx.doi.org/10.1016/j.infsof.2006.09.003

[6] T. Gorschek, P. Garre, S. B. M. Larsson, C. Wohlin, Industry evaluation of the requirements abstraction model, Requir. Eng. 12 (2007) 163–190. doi:10.1007/s00766-007-0047-z.
URL http://dl.acm.org/citation.cfm?id=1391227.1391230

[7] L. Cao, B. Ramesh, Agile requirements engineering practices: An empirical study, Software, IEEE 25 (1) (2008) 60 –67. doi:10.1109/MS.2008.1.

[8] K. Wnuk, B. Regnell, L. Karlsson, What happened to our features? visualization and understanding of scope change dynamics in a large-scale industrial setting, in: Requirements Engineering Conference, 2009. RE '09. 17th IEEE International, 2009, pp. 89 –98. doi:10.1109/RE.2009.32.

[9] C. Hood, S. Wiedemann, S. Fichtinger, U. Pautz, Requirements Management The Interface Between Requirements Development and All Other Systems Engineering Processes, Springer, Berlin, 2008. doi:10.1007/978-3-540-68476-3.
URL http://dx.doi.org/10.1007/978-3-540-68476-3

[10] J. Chen, R. Reilly, G. Lynn, The impacts of speed-to-market on new product success: the moderating effects of uncertainty, Engineering Management, IEEE Transactions on 52 (2) (2005) 199 – 212. doi:10.1109/TEM.2005.844926.

[11] C. Wohlin, M. Xie, M. Ahlgren, Reducing time to market through optimization with respect to soft factors, in: Engineering Management Conference, 1995. 'Global Engineering Management: Emerging Trends in the Asia Pacific'., Proceedings of 1995 IEEE Annual International, 1995, pp. 116 –121. doi:10.1109/IEMC.1995.523919.

[12] P. Sawyer, Packaged software: Challenges for re, in: Proceedings of the Sixth International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ 2000), 2000, pp. 137–142.

[13] D. Murphy, D. Rooney, Investing in agile: Aligning agile initiatives with enterprise goals, Cutter IT Journal 19 (2) (2006) 6 –13.

[14] J. Stephen, J. Page, J. Myers, A. Brown, D. Watson, I. Magee, System error fixing the flaws in government it, Tech. Rep. 6480524, Institute for Government, London (2011).

[15] L. Merola, The cots software obsolescence threat, in: Commercial-off-the-Shelf (COTS)-Based Software Systems, 2006. Fifth International Conference on, 2006, p. 7 pp. doi:10.1109/ICCBSS.2006.29.

[16] C. Hood, S. Wiedemann, S. Fichtinger, U. Pautz, Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes, Springer-Verlag Berin, 2008.

[17] J. Savolainen, I. Oliver, M. Mannion, H. Zuo, Transitioning from product line requirements to product line architecture, in: Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International, Vol. 1, 2005, pp. 186 – 195 Vol. 2. doi:10.1109/COMPSAC.2005.160.

[18] F. Loesch, E. Ploedereder, Restructuring variability in software product lines using concept analysis of product configurations, in: Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on, 2007, pp. 159 –170. doi:10.1109/CSMR.2007.40.

[19] M. Mannion, O. Lewis, H. Kaindl, G. Montroni, J. Wheadon, Representing requirements on generic software in an application family model, in: Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability, Springer-Verlag, London, UK, 2000, pp. 153–169.
URL http://dl.acm.org/citation.cfm?id=645546.656064

[20] T. Herald, D. Verma, C. Lubert, R. Cloutier, An obsolescence management framework for system baseline evolution perspectives through the system life cycle, Syst. Eng. 12 (2009) 1–20. doi:10.1002/sys.v12:1.
URL http://dl.acm.org/citation.cfm?id=1507335.1507337

[21] K. E. Wiegers, Software Requirements Second Edition, Microsoft Press, Redmond, WA, USA, 2003.

[22] S. Lauesen, Software Requirements – Styles and Techniques, Addison–Wesley, 2002.

[23] G. Kotonya, I. Sommerville, Requirements Engineering, John Wiley & Sons, 1998.

[24] I. Sommerville, P. Sawyer, Requirements Engineering: A Good Practice Guide, John Wiley & Sons, 1997.

[25] A. Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, John Wiley, 2009.

[26] A. Aurum, C. Wohlin, Engineering and Managing Software Requirements, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[27] IEEE, IEEE recommended practice for software requirements specifications, 830-1998, http://standards.ieee.org/findstds/standard/830-1998.html (September 1997).

[28] S. E. Institute, Capability maturity model integration (cmmi), version 1.3, http://www.sei.cmu.edu/cmmi/solutions/info-center.cfm (last visited, December 2011).

[29] C. Hood, S. Wiedemann, S. Fichtinger, U. Pautz, Change management interface, in: Requirements Management, Springer Berlin Heidelberg, 2008, pp. 175–191.

[30] S. Robertson, J. Robertson, Mastering the requirements process, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[31] C. Iacovou, A. Dexter, Turning around runaway information technology projects, Engineering Management Review, IEEE 32 (4) (2004) 97 –112. doi:10.1109/EMR.2004.25141.

[32] I. Legodi, M.-L. Barry, The current challenges and status of risk management in enterprise data warehouse projects in south africa, in: Technology Management for Global Economic Growth (PICMET), 2010 Proceedings of PICMET '10:, 2010, pp. 1 –5.

[33] T. DeMarco, T. Lister, Risk management during requirements, IEEE Software 20 (5) (2003) 99–101.

[34] IEEE Computer Society, Software Engineering Body of Knowledge (SWEBOK), Angela Burgess, EUA, 2004.
URL http://www.swebok.org/

[35] D. Zowghi, N. Nurmuliani, A study of the impact of requirements volatility on software project performance, Asia-Pacific Software Engineering Conference 0 (2002) 3. doi:http://doi.ieeecomputersociety.org/10.1109/APSEC.2002.1182970.

[36] S. Harker, K. Eason, J. Dobson, The change and evolution of requirements as a challenge to the practice of software engineering, in: Requirements Engineering, 1993., Proceedings of IEEE International Symposium on, 1993, pp. 266 –272. doi:10.1109/ISRE.1993.324847.

[37] S. McGee, D. Greer, A software requirements change source taxonomy, in: Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on, 2009, pp. 51 –58. doi:10.1109/ICSEA.2009.17.

[38] S. Easterbrook, J. Singer, M.-A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: F. Shull, J. Singer, D. I. K. Sjberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer London, 2008, pp. 285–311.

[39] J. Singer, S. E. Sim, T. C. Lethbridge, Software engineering data collection for field studies, in: F. Shull, J. Singer, D. I. K. Sjberg (Eds.),

Guide to Advanced Empirical Software Engineering, Springer London, 2008, pp. 9–34.

[40] C. Dawson, Projects in Computing and Information Systems: A Student's Guide, Addison Wesley, 2005.

[41] R. A. P. L. M. Rea, Designing and Conducting Survey Research: A Comprehensive Guide, Jossey-Bass, San Francisco, CA, 94103-1741, 1005.

[42] K. Wnuk, The survey questionnaire, http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/AppendixB_SurveyQuestions.pdf (last visited, December 2011).

[43] Wikipedia, Likert scale, http://en.wikipedia.org/wiki/Likert_scale (last visited, December 2011).

[44] B. Regnell, R. B. Svensson, K. Wnuk, Can we beat the complexity of very large-scale requirements engineering?, in: Proceedings of the 14th international conference on Requirements Engineering: Foundation for Software Quality, REFSQ '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 123–128.

[45] S. Monkey, Survey monkey webpage, http://www.surveymonkey.net (last visited, December 2011).

[46] Linkedin, The linkedin website, http://www.linkedin.com/ (last visited, December 2011).

[47] K. Wnuk, The full list of mailing lists can be accessed at, http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/ListOfDiscussionGroups.pdf (last visited, December 2011).

[48] K. Wnuk, The full list of tool vendors can be accessed at, http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/ListOfVendors.pdf (last visited, December 2011).

[49] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering: an introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[50] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Proceeding of the 33rd International Conference on Software Engineering, ICSE '11, ACM, New York, NY, USA, 2011, pp. 1–10. doi:http://doi.acm.org/10.1145/1985793.1985795.
URL http://doi.acm.org/10.1145/1985793.1985795

[51] S. Nakagawa, A farewell to Bonferroni: the problems of low statistical power and publication bias, Behavioral Ecology 15 (6) (2004) 1044–1045.

[52] T. C. Lethbridge, S. E. Sim, J. Singer, Studying software engineers: Data collection techniques for software field studies, Empirical Software Engineering 10 (2005) 311–341, 10.1007/s10664-005-1290-x.
URL http://dx.doi.org/10.1007/s10664-005-1290-x

[53] S. Siegel, N. J. Castellan, Nonparametric statistics for the behavioral sciences, 2nd Edition, McGraw-Hill, 1998.

[54] K. Wnuk, The appendix with analysis can be accessed at, http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/AppendixA_Analysis.pdf (last visited, December 2011).

[55] K. Wnuk, The full list of countries can be obtained at, http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/COUNTRIES.pdf (last visited, December 2011).

[56] IBM, The description of the method can be found at, http://www-01.ibm.com/software/awdtools/rup/ (last visited, December 2011).

[57] B. Berenbach, D. J. Paulish, J. Kazmeier, A. Rudorfer, Software & Systems Requirements Engineering: In Practice, Pearson Education Inc., 2009.

[58] A. Quispe, M. Marques, L. Silvestre, S. Ochoa, R. Robbes, Requirements engineering practices in very small software enterprises: A diagnostic study, in: Chilean Computer Science Society (SCCC), 2010 XXIX International Conference of the, 2010, pp. 81 –87. doi:10.1109/SCCC.2010.35.

[59] M. Bano, N. Ikram, Issues and challenges of requirement engineering in service oriented software development, in: Software Engineering Advances (ICSEA), 2010 Fifth International Conference on, 2010, pp. 64 –69. doi:10.1109/ICSEA.2010.17.

[60] M. Kossmann, A. Gillies, M. Odeh, S. Watts, Ontology-driven requirements engineering with reference to the aerospace industry, in: Applications of Digital Information and Web Technologies, 2009. ICADIWT '09. Second International Conference on the, 2009, pp. 95 –103. doi:10.1109/ICADIWT.2009.5273953.

[61] I. Sommerville, Software Engineering, Addison–Wesley, 2007.

[62] W. Curtis, H. Krasner, V. Shen, N. Iscoe, On building software process models under the lamppost, in: Proceedings of the 9th international conference on Software Engineering (ICSE 1987), 1987, pp. 96–103.

[63] T. Gorschek, A. M. Davis, Requirements engineering: In search of the dependent variables, Inf. Softw. Technol. 50 (2008) 67–75. doi:10.1016/j.infsof.2007.10.003.
URL http://dl.acm.org/citation.cfm?id=1324618.1324710

[64] S. Easterbrook, What is requirements engineering?, http://www.cs.toronto.edu/~sme/papers/2004/FoRE-chapter01-v7.pdf (July 2004).

[65] T. Gorschek, C. Wohlin, Requirements abstraction model, Requir. Eng. 11 (2005) 79–101. doi:10.1007/s00766-005-0020-7.
URL http://dl.acm.org/citation.cfm?id=1107677.1107682

[66] J. Kabbedijk, B. R. K. Wnuk, S. Brinkkemper, What decision characteristics influence decision making in market-driven large-scale software product line development?, in: Product Line Requirements Engineering and Quality 2010, 2010, pp. 42 –53.

[67] R. Kohl, Changes in the requirements engineering processes for cots-based systems, Requirements Engineering, IEEE International Conference on 0 (2001) 0271. doi:http://doi.ieeecomputersociety.org/10.1109/ISRE.2001.948575.

[68] H. Holmstrom, E. O. Conchuir, P. J. Agerfalk, B. Fitzgerald, Global software development challenges: A case study on temporal, geographical and socio-cultural distance, in: Global Software Engineering, 2006. ICGSE '06. International Conference on, 2006, pp. 3 –11. doi:10.1109/ICGSE.2006.261210.

[69] S. Buhne, G. Halmans, K. Pohl, M. Weber, H. Kleinwechter, T. Wierczoch, Defining requirements at different levels of abstraction, in: Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International, 2004, pp. 346 – 347. doi:10.1109/ICRE.2004.1335694.

[70] J. Natt Och Dag, T. Thelin, B. Regnell, An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development, Empirical Softw. Engg. 11 (2006) 303–329. doi:10.1007/s10664-006-6405-5.
URL http://dl.acm.org/citation.cfm?id=1120556.1120562

[71] D. Šmite, C. Wohlin, T. Gorschek, R. Feldt, Empirical evidence in global software engineering: a systematic review, Empirical Softw. Engg. 15 (2010) 91–118. doi:http://dx.doi.org/10.1007/s10664-009-9123-y.
URL http://dx.doi.org/10.1007/s10664-009-9123-y

[72] W. Anderson, P. J. Boxer, L. Brownsword, An examination of a structural modeling risk probe technique, Tech. Rep. CMU/SEI-2006-SR-017, Software Engineering Institute, Carnegie Mellon University (2008).

[73] M. Bekker, H. Steyn, The impact of project governance principles on project performance, in: Management of Engineering Technology, 2008. PICMET 2008. Portland International Conference on, 2008, pp. 1324 –1330. doi:10.1109/PICMET.2008.4599744.

[74] B. Boehm, Software risk management: principles and practices, Software, IEEE 8 (1) (1991) 32 –41. doi:10.1109/52.62930.

[75] K. Pohl, Requirements Engineering: Fundamentals, Principles, and Techniques, 1st Edition, Springer Publishing Company, Incorporated, 2010.

Figure1

Click here to download high resolution image

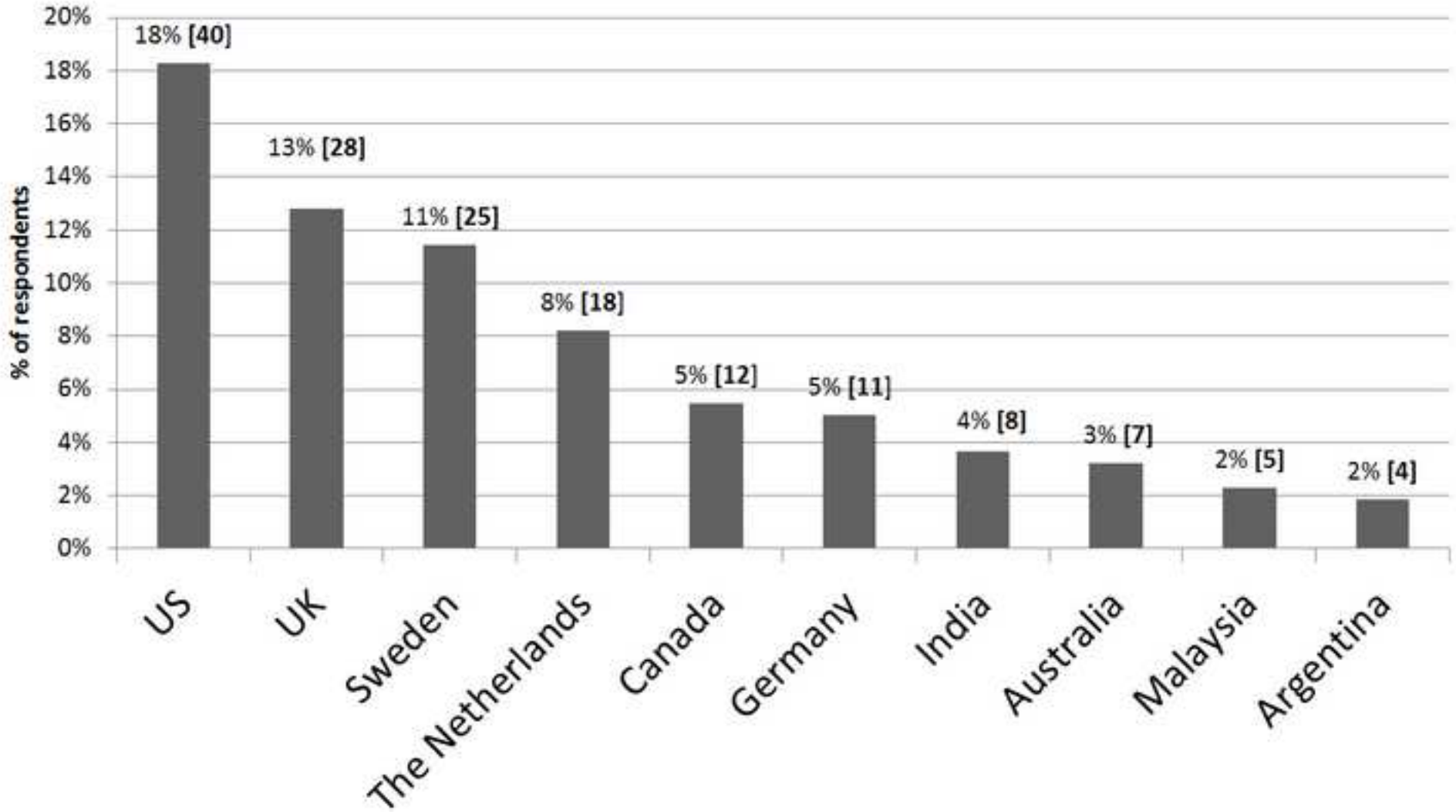

Figure2
Click here to download high resolution image

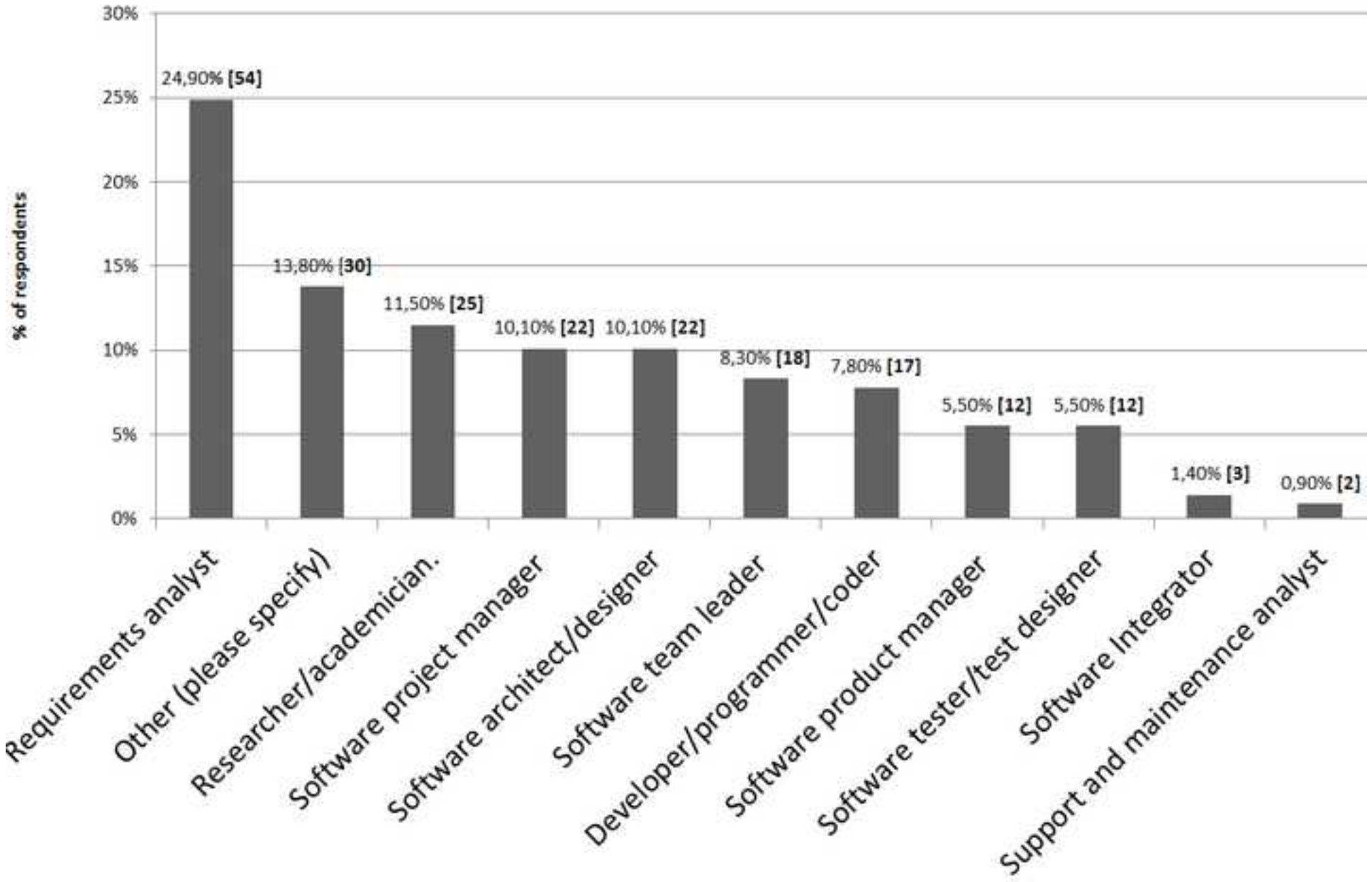

Figure3

Click here to download high resolution image

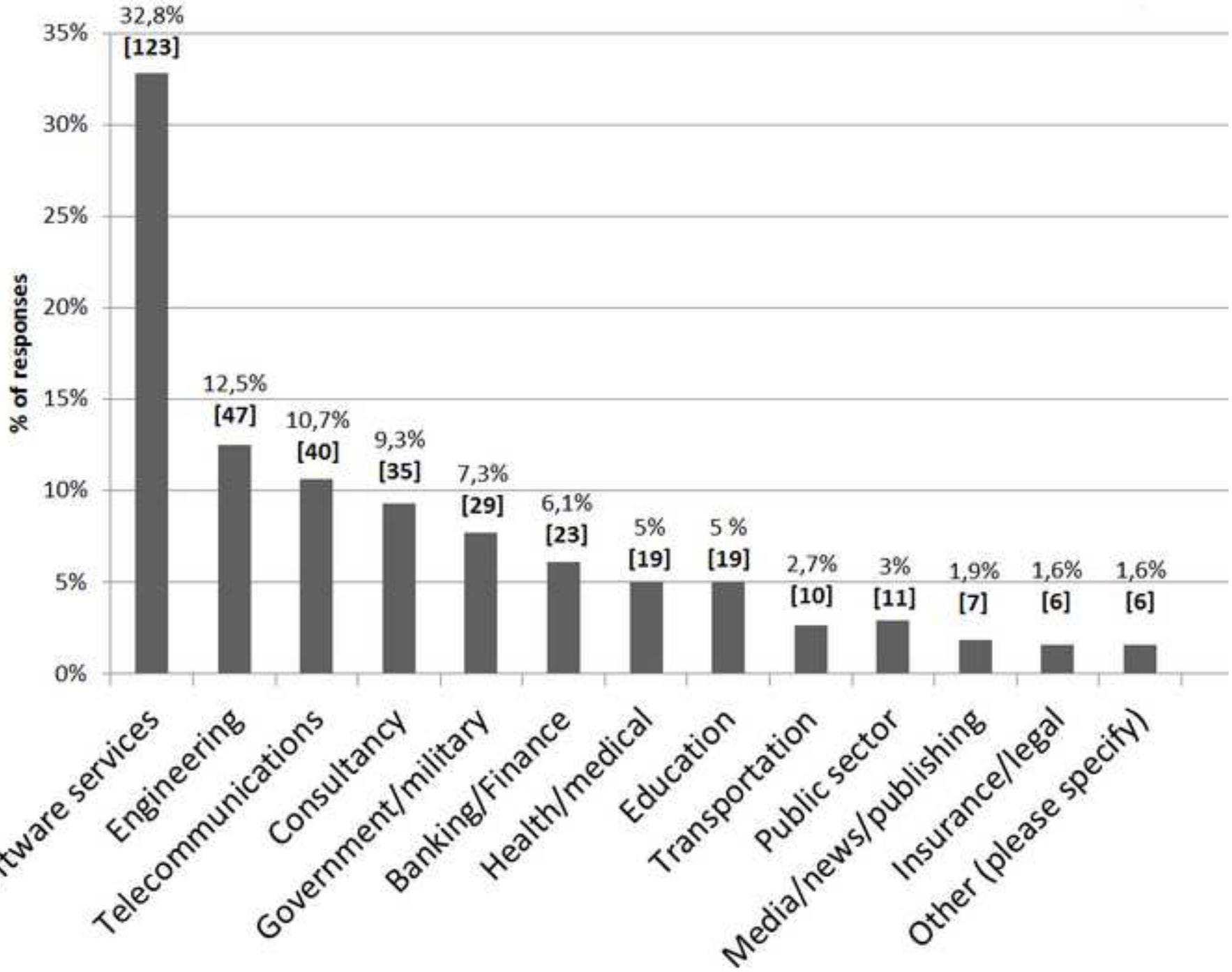

Figure4

Click here to download high resolution image

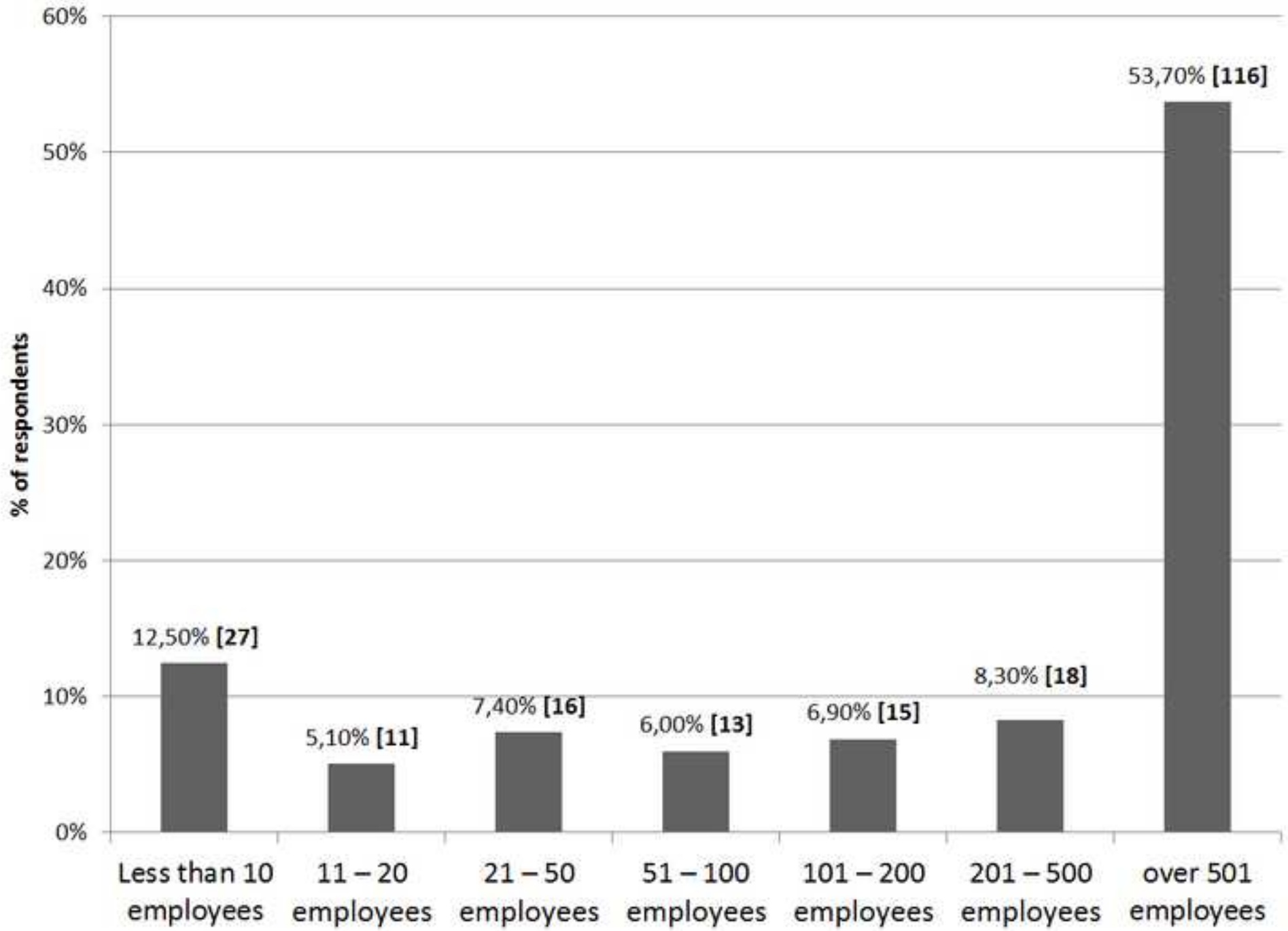

Figure5
Click here to download high resolution image

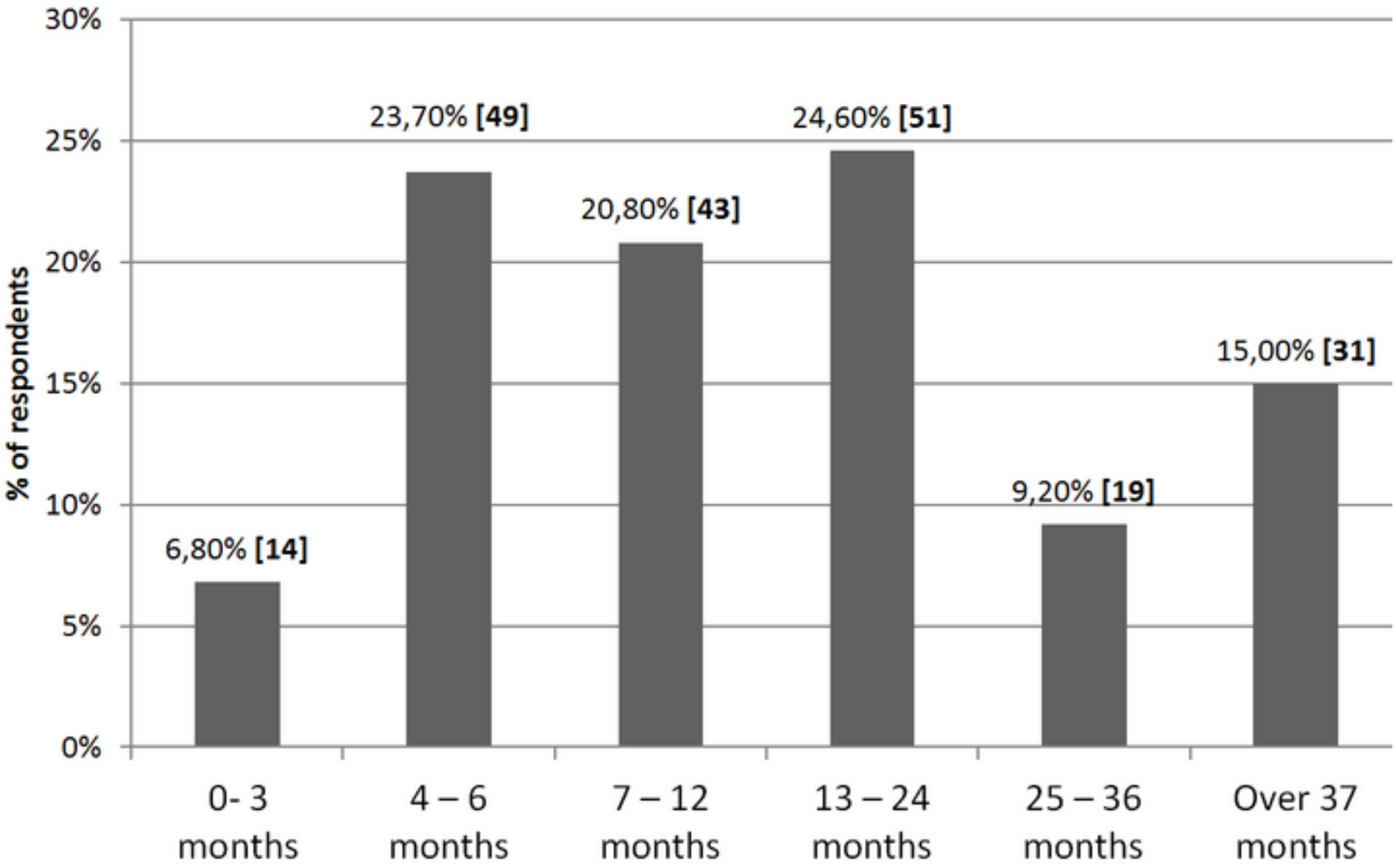

Figure6

Click here to download high resolution image

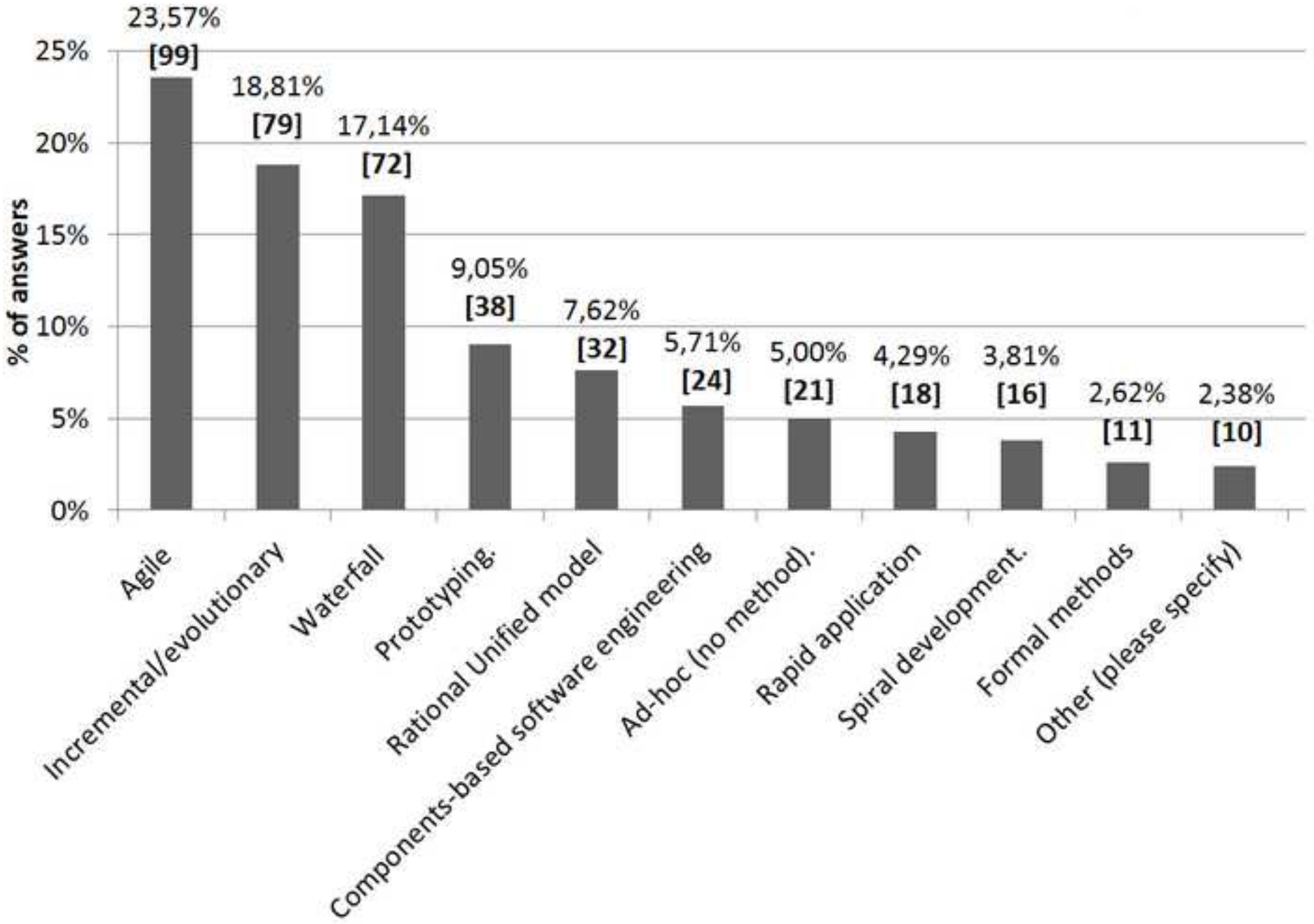

Figure7
Click here to download high resolution image

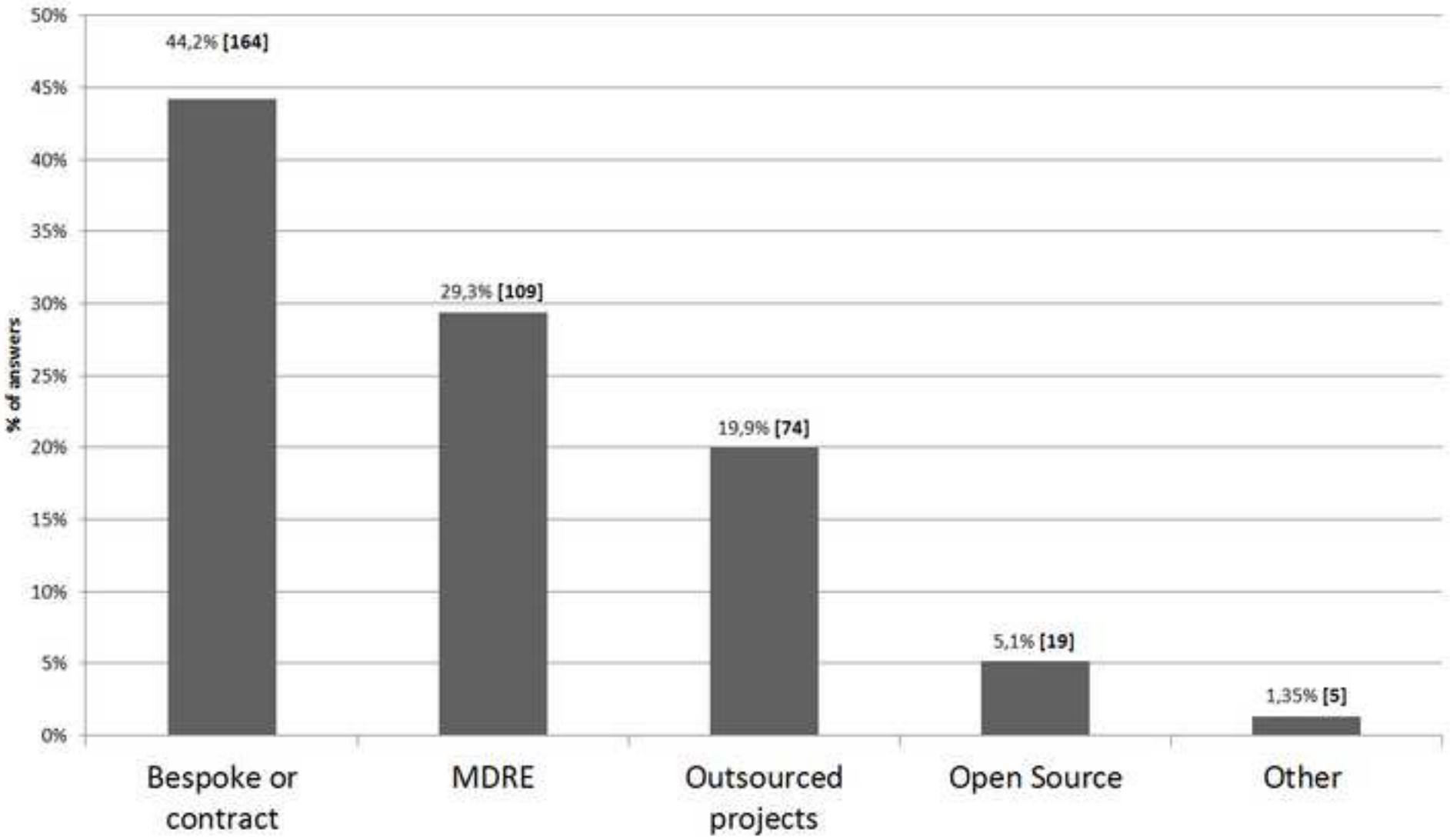

Figure8
Click here to download high resolution image

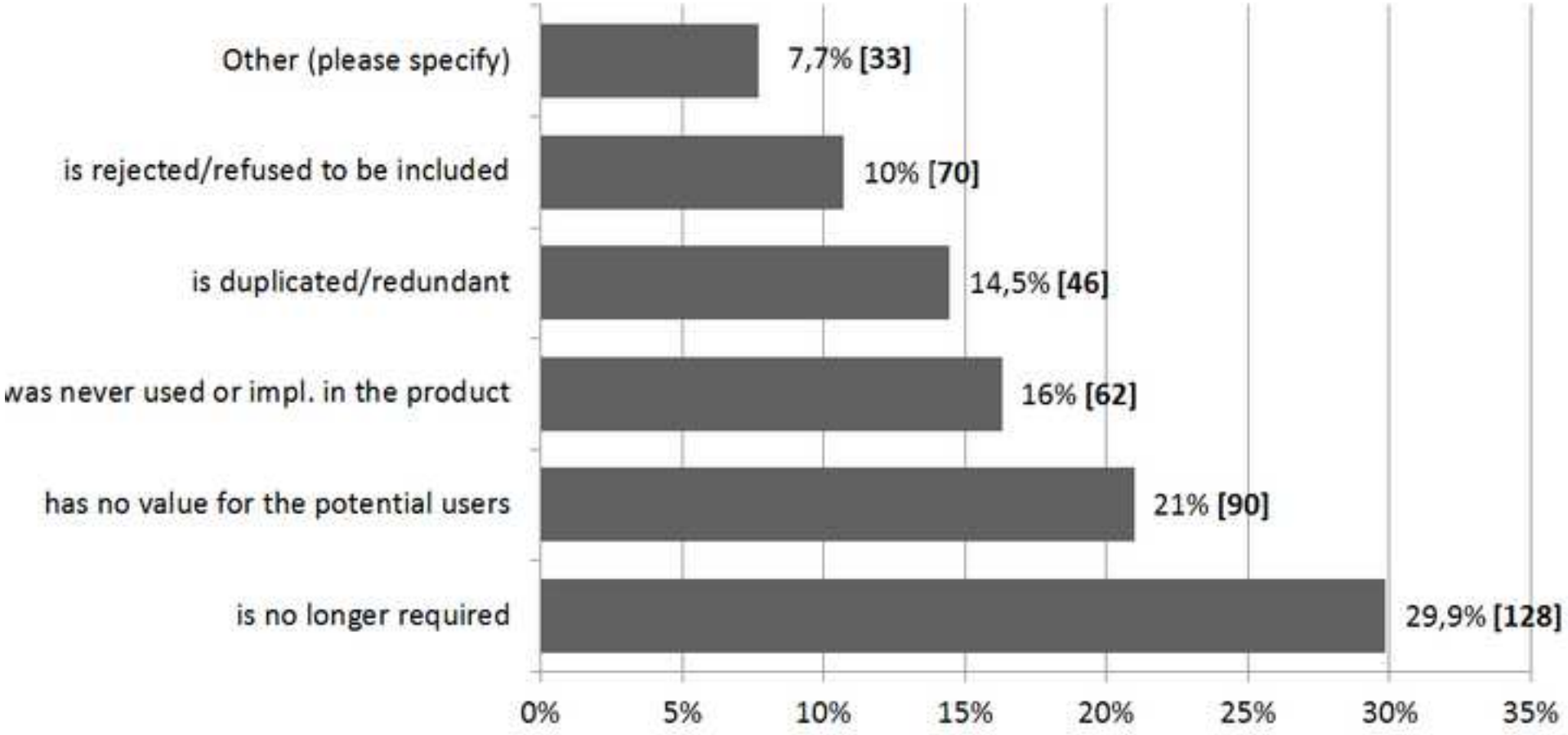

Figure9
Click here to download high resolution image

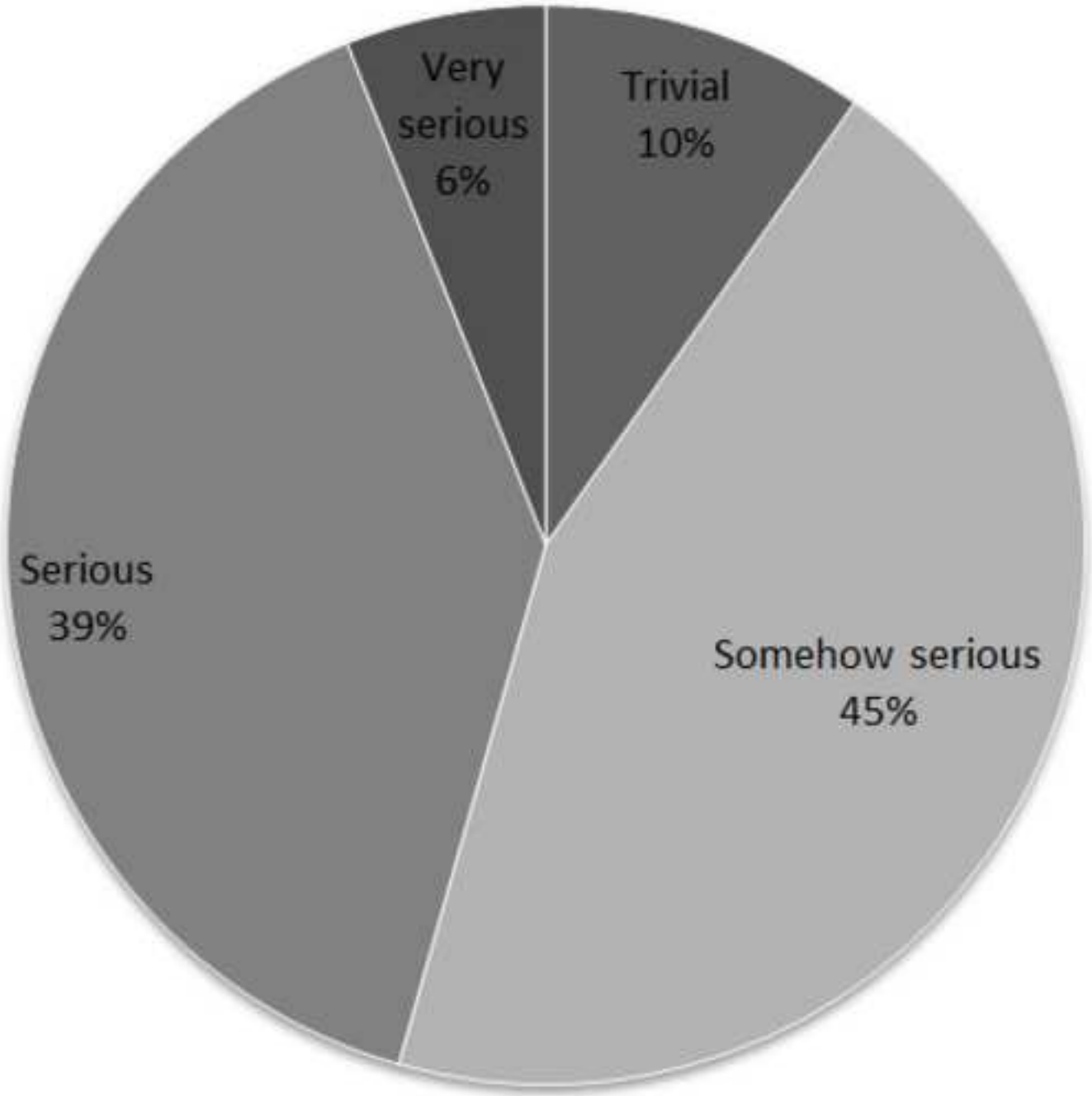

Figure10
Click here to download high resolution image

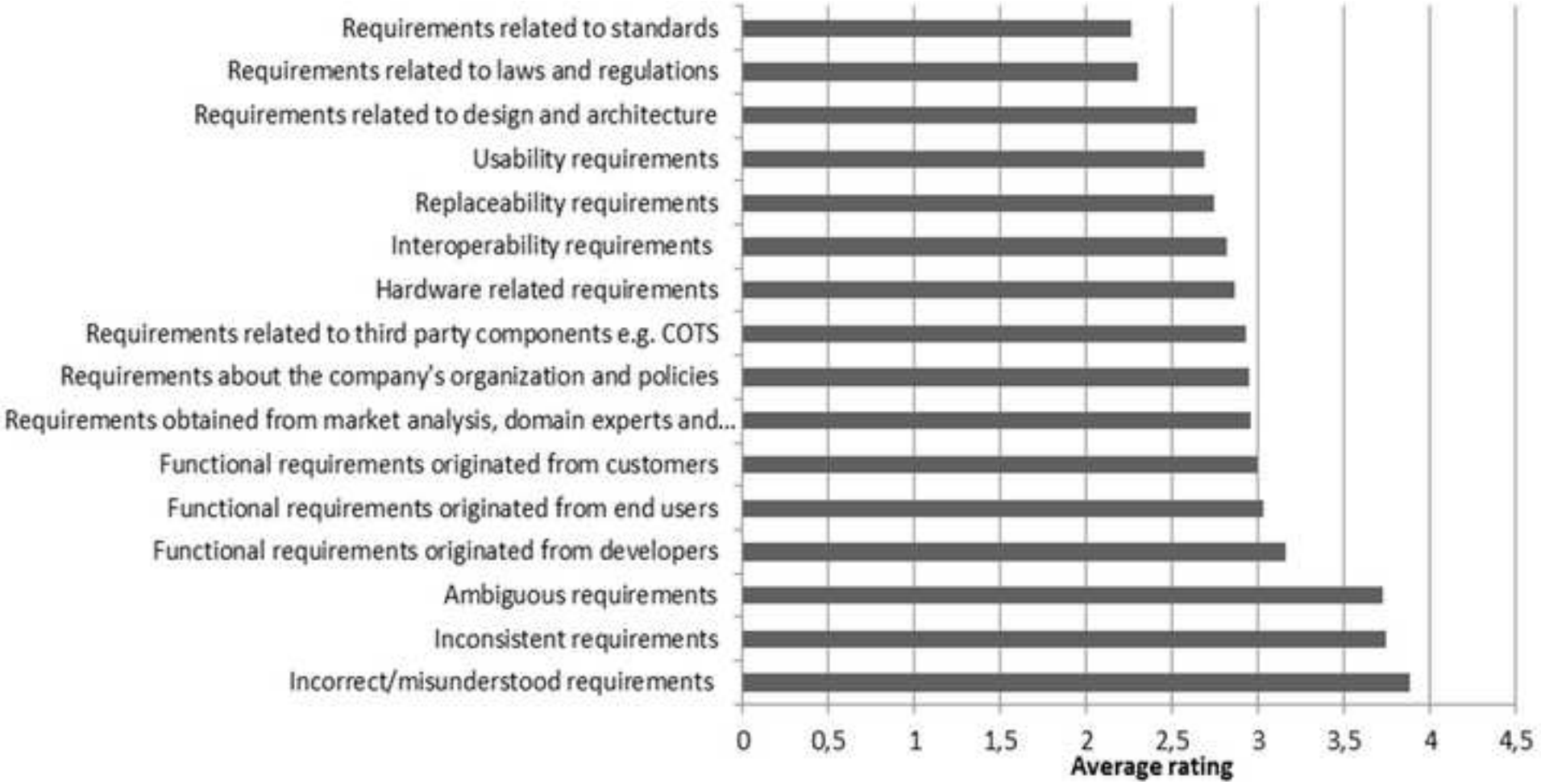

Figure11
Click here to download high resolution image

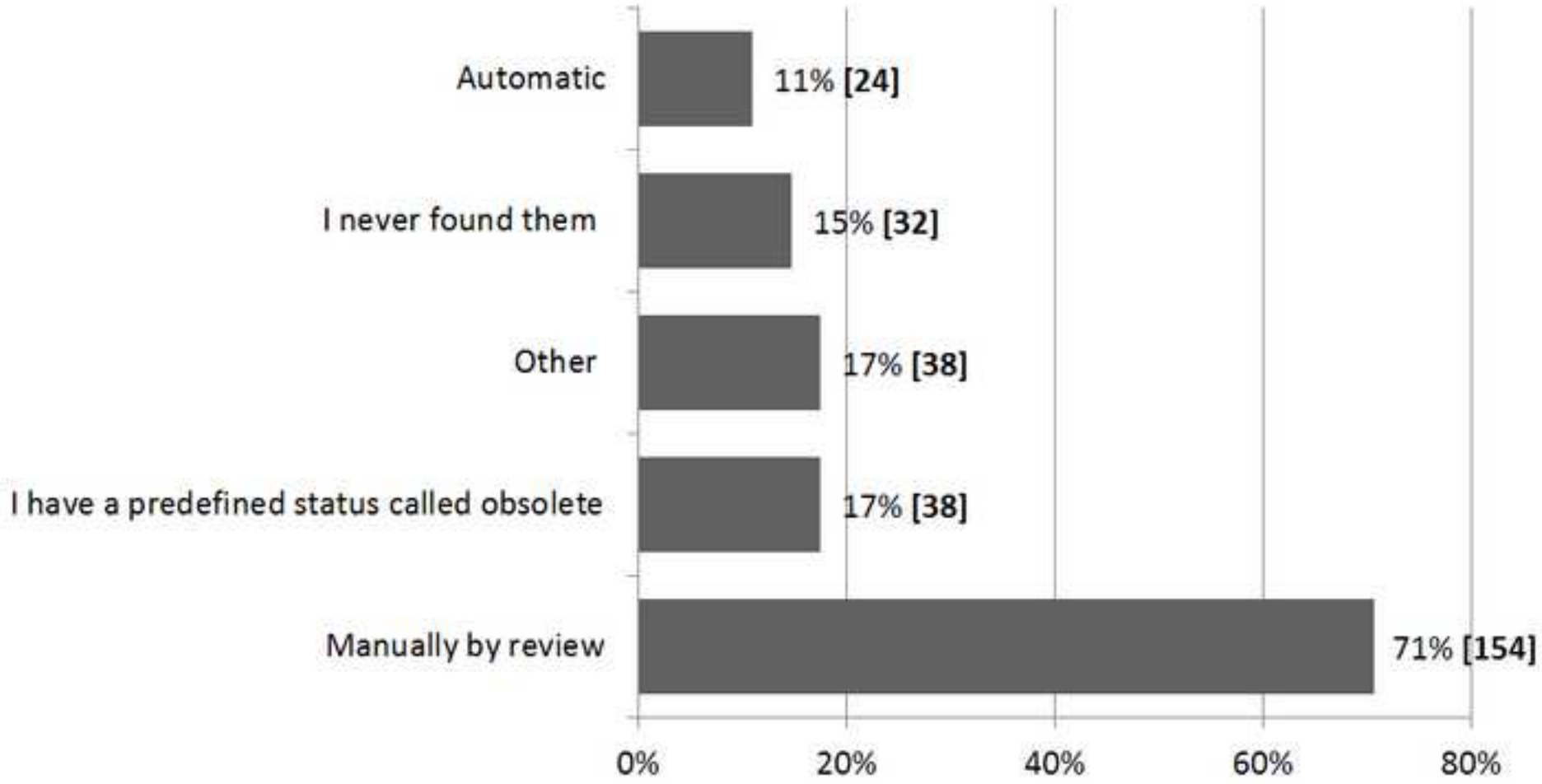

Figure12
Click here to download high resolution image

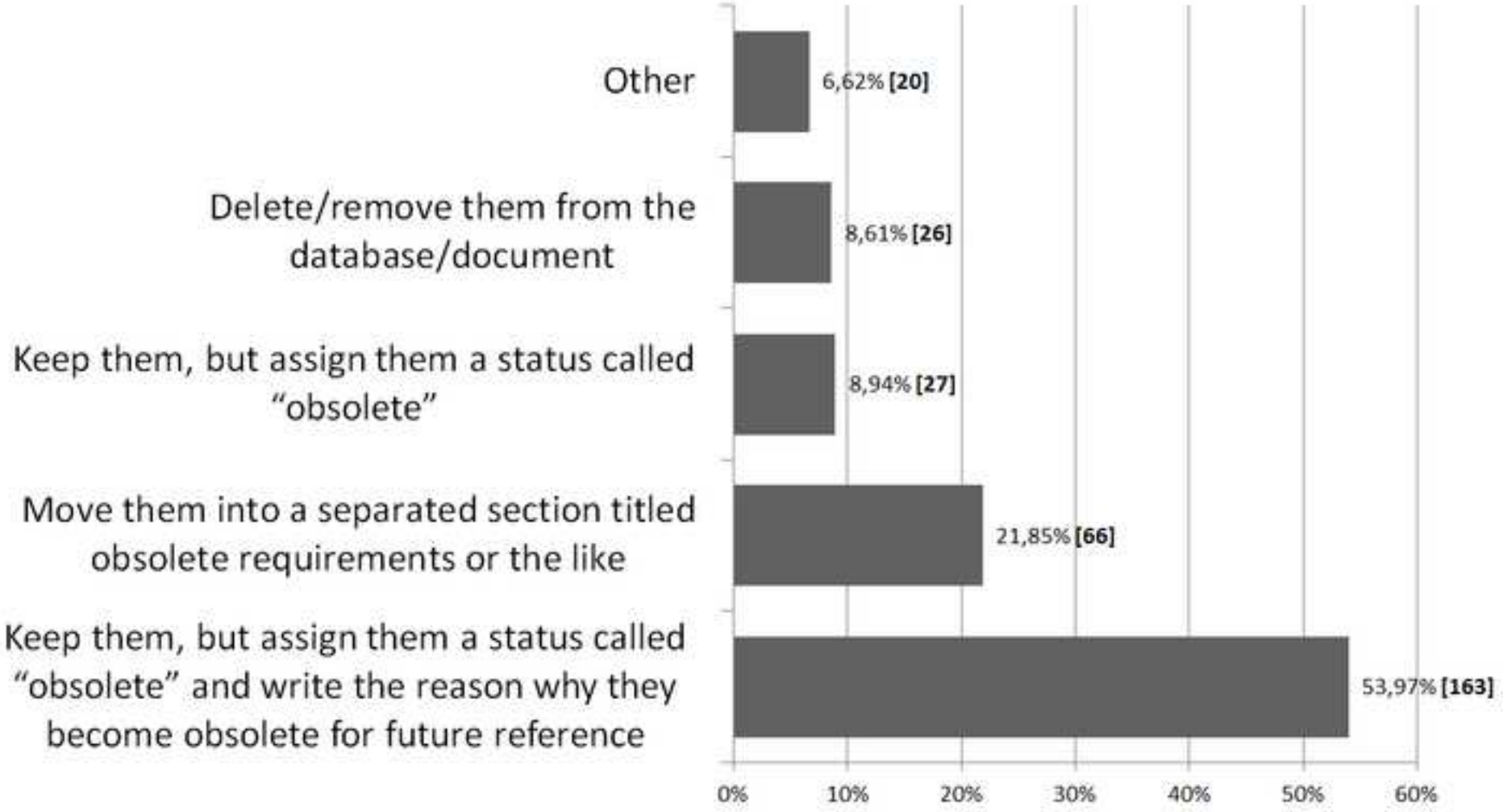

Figure13
Click here to download high resolution image